ENCYCLOPEDIA FOR THE TRS-80*

A library of useful information
for your TRS-80*

Business
Education
Games
Graphics
Hardware
Home Applications
Interface
Tutorial
Utility

VOLUME 3

ENCYCLOPEDIA for the TRS-80*

ENCYCLOPEDIA for the TRS-80*

VOLUME 3

wayne GREEN INC.
PETERBOROUGH NH 03458

*Trademark of Tandy Corp.

FIRST EDITION
FIRST PRINTING OCTOBER 1981

Printing in the United States of America

Edited by Kate Comiskey, Chris Crocker,
Nan McCarthy, Katherine Putnam
Technical Assistance by Jake Commander,
Kenniston Lord Jr, CPD, Dennis Thurlow
Illustrations Howard Happ
Typeset by Karen Stewart

FOREWORD

The Biggest Difference

There are lots of arguments about which computer is the best. The answer to this question lies not in which hardware is best. That is really irrelevant, when you understand the field. The major value of any computer lies in the software and the information available for it. Hence this encyclopedia.

The TRS-80 is by no means the best computer on the market as far as its hardware is concerned, but with the support of *80 Microcomputing* magazine and this encyclopedia series, you have an almost unlimited source of information on how to use your computer—and of programs. With this information source the TRS-80 is by far the most valuable computer system ever built. No other computer, at any price, has anything approaching this amount of user information and programs available.

Most encyclopedias try to freeze everything at one time and are thus able to divide the material up alphabetically. This is a new kind of encyclopedia —a living one—with each new volume keeping you up to date on the very latest information on using your computer and the newest of programs.

Your computer can be a fantastic teaching device, a simulator, a way to play all sorts of fascinating games, a business aid, a scientific instrument, a control unit for machinery. . . . It is one of the most flexible gadgets ever invented. All of these applications are possible *if* you have the information and the programs. This encyclopedia will give you these.

To get the best use of your TRS-80, don't miss a single volume of the *Encyclopedia for the TRS-80.*

WAYNE GREEN
Publisher

CONTENTS

Please note: Before typing in any listing in this book, see Appendix A.

contents

BUSINESS

Flex/Form

Inventory

Flex/Form

by Jon Mark O'Connor

Whether you're writing two or three in-house memos, 150 letters to business acquaintances, or inviting 500 friends to your house for coffee and doughnuts, Flex/Form will take the drudgery out of the job. Each letter will be personalized. All you have to do is sign the letters and put them in envelopes.

A 16K machine will allow between 125-150 names and 48K will give you access to 500. The program works well on disk.

No matter how many lines of text, the letter will appear neatly in the middle of page. Since this is more of an informal approach to form-letter writing, I chose not to bother with right justification.

The program has the ability to take any input—up to 30 characters—and search through the data. Any data statement having that combination of letters will be sent to either the screen or the printer. Using the two example data statements in line 1000-1001 we can retrieve both names from the list by simply typing 80. As soon as 80 is matched, then each name, address, city, state, etc. will be printed. In-house memos are not set up exactly as in 1001, but notice there are only three pieces of data in that line followed by a comma. If the list contained everyone who worked at Instant Software and *80 Microcomputing* and you wanted only the editorial staff at a meeting, then you would only have to type edit or editorial. If you wanted to invite only the folks from *80*, it might be better to use the fourth piece of data and insert a code such as E-80 and likewise E-IS. As long as the code is not a combination that could be in a name or address then you can use it. Admittedly, the search is slow, but considering what the computer has to go through, one second per data line is not all that bad. I recommend that a popular list of names appear high in your data list. Do not attempt to mix and match names and addresses in your input. It will not work.

Since this is an informal letter-writer, I chose to allow a maximum of only 10 lines of text. When typing in this program you may change the availability of text lines by changing line 260 to meet your needs. The variable KO is the text counter and also the variable that regulates the printer spacing. In line 320 notice the expression : FOR LP=1 TO (32-KO)/2. This controls the spaces from the top of the page to the first line of your letterhead. An identical expression is in line 360. If you don't want your pages to be approximately 11 1/2 inches long, then play around with the number 32 in each line. You could add another input line that controls the length of the paper. This is entirely up to you, and since most

memo sheets would not be 11 1/2 inches, this may be significant for you to experiment with. Also, the number 32 is the actual overhead of already-used lines (letterhead, salutation, line space, etc.). If you want to add a second or third page of text you will have to consider the overhead and the fact that there are normally 66 printer lines for an 11 1/2" page. A simple GOSUB routine will handle this for you.

The Program

The first thing you have to do is type in your name and address, so hit 1. Develop the habit of using an opening quote mark. If you inadvertently insert a comma into any input line, you will get an ?EXTRA IGNORED error. Though you can retype the line by hitting 9876, this is a bother, so use the quote mark. Don't type in the date now. Briefly, a sample of the letter with only the above input will be displayed. This is not the way it will be spaced in your letter.

The screen will show:

```
        9999 TO END : ERROR - HIT 9876
    LINE # 1
    ? _
```

Hit 9999 for now, and you should be back at the menu. You have another option at this point. If you want to bother with a data tape, do it. Hit 6 and follow the instructions. Use a clean tape for data tapes. Sending this data to the tape takes all of six seconds. If you're a fast typist and don't mind typing in your name and address each time, you can delete lines 290-310. Change the menu and delete the references to these lines in line 40. This will greatly increase available memory for data lines. Also change line 390 to read:

```
390 CLS: LIST 1000 - 29999
```

Delete line 400, 1000, and 1001, but I would keep all of the above in the program until you're familiar with it.

The only operation that will take you out of the program is updating the data list (4). Insert a couple of data lines beginning with 1010. All data statements must have the equivalent of four pieces of information. Any piece that has a comma or colon must have opening and closing quote marks. Since there are four READs, each line must have commas (example in line 1001) in case there are not four pieces. If you want to add data pieces to each data line, change line 100 accordingly. Insert new READ statements and similar MID$ (x, P, M) statements. If you use extra data pieces and use them merely as a location for the search, they need not be printed; therefore, line 100 should be the only line changed. If this new data is to be included in the letter, I leave it up to you to alter the rest of the program.

Having inserted your few data lines for a test, now write a letter. Since you have left the program, you're going to have to call up the data tape and

retrieve the information. Read the instructions after hitting 5. You should be back at the menu after retrieval. Hit 1. Notice that the program knows you have retrieved the information and from now on all you have to do is type in the date and the letter. You can type the date "December 12, 1981" with the quotes.

You will see the letterhead and P.S. again. Then you are ready to type a letter. Make it a short two-liner for now. Type only to the arrow and hit ENTER. Now hit 9876. The last line you typed will remain on the screen to help maintain continuity of thought. As soon as you retype the line and hit ENTER, that line will be replaced.

When you have typed in your two-liner, hit 9999 to return to the menu. Next hit 3 to see your letter. I have mentioned this many times, but don't forget the opening quote mark (disk users may use LINE INPUT for all INPUT statements to avoid this nuisance). If your text doesn't look exactly the way you think you typed it in, then you have probably inserted a comma. Hit 1 again. Retype the date. When you're back at the letter-writing section, you'll notice the first line of your text appears at the top of the screen. Slowly hit the ENTER key until the offending line appears. (You'll have to hit ENTER for the second line twice.) Now type 9876 to change that line and then continue to write the rest of the letter. As soon as 9876 is entered, all lines below the mistake line are erased from memory.

Now you have a finished letter and are back at the menu. Hit 2 for the search. By hitting either the down- or right-arrows we will see the complete data list. Hit ENTER. The screen will display:

ENTER FOR LIST OF ALL ";A;"S OR HIT 7777 FOR LETTER?_

Hit ENTER and a maximum of four of the names you have inserted will appear. Hit ENTER to continue the list. We can now send a letter to one of the names on the list or return to the menu by hitting 9999. Enter a name or just a character in one of the names displayed and then hit 7777 for the letter. Within moments a letter will be printed. And that's all there is to it.

---

56 EUSTIS PARKWAY
WATERVILLE, MAINE 04901
12/12/80

WAYNE GREEN
80 MICROCOMPUTING
PETERBOROUGH, NEW HAMPSHIRE 03458

DEAR WAYNE,

YOU ARE CORDIALLY INVITED TO THE MAINE/FLO OPEN HOUSE

continued

ON JANUARY 16, 1980. WE WILL BE DISPLAYING SOME TRULY REMARKABLE STATE OF THE ART HARDWARE FOR THE TRS-80. WE FEEL THE BEST WAY TO SHOW OFF OUR FINE LINE OF PRODUCTS IS TO ALLOW OUR PROSPECTIVE CUSTOMERS A HANDS-ON DEMONSTRATION OF THEIR CAPABILITIES.

PLEASE COME TO OUR PARTY. THE DOORS WILL BE OPEN TO A SELECT FEW FROM 1 P.M. TO 10 P.M. AND WE HOPE YOU CAN BE PART OF THE FESTIVITIES.

SINCERELY,

JON MARK O'CONNOR

P.S. CALL ME AT 1-207-555-1212

Figure 1. *Sample letter*

FROM THE PUBLISHER'S DESK
12/12/80

DAVID SYLVER
EDITORIAL STAFF
80 MICROCOMPUTING

DEAR DAVID,

I AM IN DOUBT REGARDING THE NEW LAYOUT OF OUR MAGAZINE. I WISH TO SPEAK TO EACH MEMBER OF THE EDITORIAL STAFF AT FOUR O'CLOCK THIS AFTERNOON SO THAT WE CAN DISCUSS SOME MUCH NEEDED CHANGES.

SINCERELY,

WAYNE

P.S. CALL ME AT EXT. 345

Figure 2. *Sample letter*

Program Listing

```
10 CLS :
   DEF @477,"FLEX / FORM":
   PRINT :
   PRINT TAB(26)"JON MARK O'CONNOR":
   PRINT TAB(26)"56 EUSTIS PARKWAY":
   PRINT TAB(26)"WATERVILLE, MAINE ":
   FOR T = 1 TO 500:
    NEXT :
   CLEAR 800:
   DIM W$(30):
   DEFSTR A - F,I,Q,R,T,U:
   DEFINT G,K,M,O,P:
   KO = 0:
   W = 0:
   P9 = 0:
   A9 = CHR$(229)
20 CLS :
   LK = 1:
   PRINT @0,"MEMORY LEFT"; MEM :
   PRINT @320,
30 PRINT TAB(18)"ENTER NEW LETTER"; TAB(42)"<1>";A9;"SEARCH FOR NAM
   E"; TAB(42)"<2>";A9;"SEE LETETER"; TAB(42)"<3>";A9;"UPDATE DATA
   LIST"; TAB(42)"<4>";A9;"GET ADDRESS FROM TAPE"; TAB(42)"<5>";A9;
   "SEND ADDRESS TO TAPE"; TAB(42)"<6>"
40 U = INKEY$:
   IF U > CHR$(48) AND U < CHR$(55)
    THEN
     ON VAL(U) GOTO 160,60,50,390,310,290:
    ELSE
     40
50 CLS :
   PRINT @128,:
   FOR G = 0 TO KO + 1:
    PRINT W$(G):
    NEXT :
   INPUT "HIT ENTER";L:
   GOTO 20
60 CLS :
   PRINT @980,:
   GOTO 80
70 PRINT CHR$(143); STRING$(23,140);" LIST COMPLETE "; STRING$(23,1
   40); CHR$(143):
   LK = 1
80 X = 0:
   V = 960:
   PRINT "INPUT NAME, ADDRESS, OR PHONE # <9999 FOR MENU>    ";
   CHR$(92):
   PRINT TAB(10):
   INPUT "SEARCH ->";A:
   IF LEN(A) > 30
    THEN
     PRINT @960, CHR$(255) CHR$(255):
     PRINT TAB(10)"INPUT AGAIN. LIMIT TO 30 CHARACTERS. ":
     GOTO 80:
    ELSE
     IF A = "9999"
      THEN
       20:
      ELSE
       CLS
90 PRINT @960,"ENTER FOR LIST OF ALL ";A;"'S  OR HIT  7777 FOR LETT
   ER ";:
   INPUT X:
   PRINT @346,"SEARCHING":
   PRINT @896, CHR$(255):
   M = LEN(A):
   O = 1:
   RESTORE
```

Program continued

```
100 READ B:
    READ C:
    READ D:
    READ E:
    IF B = "*"
     THEN
      70:
     ELSE
      FOR P = 1 TO 30:
       IF A = MID$(B,P,M)
        THEN
         110:
        ELSE
         IF A = MID$(C,P,M)
          THEN
           110:
          ELSE
           IF A = MID$(D,P,M)
            THEN
             110:
            ELSE
             IF A = MID$(E,P,M)
              THEN
               110:
              ELSE
               NEXT :
              IF P > 30
               THEN
                100
110 IF X = 7777 GOSUB 320:
    GOTO 100:
     ELSE
      IF LK = > 2
       THEN
        PRINT STRING$(63,179); CHR$(128);:
       ELSE
        PRINT STRING$(63,176); CHR$(128);
120 PRINT @V, CHR$(191);B;:
    FOR G = LEN(B) + (V + 2) TO 35 + V:
     PRINT @G,".";:
     NEXT :
    PRINT TAB(37)"TELEPHONE ";:
    IF E = ""
     THEN
      PRINT " <NOT LISTED>";:
      GOTO 130:
     ELSE
      PRINT E;
130 PRINT TAB(62) CHR$(191):
    PRINT CHR$(191);C; TAB(62) CHR$(191):
    PRINT CHR$(191);D; TAB(50)"<";LK;">"; TAB(62) CHR$(191):
    O = O + 1:
    LK = LK + 1:
    IF O = 5 GOSUB 150:
    GOTO 100:
     ELSE
      100
140 O = O + 1:
    GOTO 100
150 PRINT TAB(25):
    INPUT "HIT ENTER";L:
    PRINT @896, CHR$(191); TAB(62); CHR$(191):
    O = 1:
    RETURN
160 IF P9 = 99
     THEN
      W = 0:
      GOTO 210:
     ELSE
      CLS :
```

```
     PRINT @960,"THE DATA SUPPLIED INDICATES THAT YOU HAVE NOT TYPE
     D IN SOME      IMPORTANT INFORMATION. PLEASE DO SO NOW.    IF YO
     U NOTICE AN      ERROR ON THE PREVIOUS LINE HIT  9876 TO RETURN
      TO THAT LINE."
170 PRINT :
    INPUT "TYPE IN YOUR STREET ADDRESS";Q
180 INPUT "YOUR CITY STATE AND ZIP  (PLACE IN QUOTES)";R:
    IF R = "9876"
     THEN
      170
190 INPUT "TYPE IN  YOUR TELEPHONE #";F:
    IF F = "9876"
     THEN
      180
200 INPUT "TYPE IN YOUR NAME";I:
    IF I = "9876"
     THEN
      190:
     ELSE
      P9 = 99:
      GOTO 220
210 CLS :
    PRINT @960,"YOU HAVE ENTERED INFO FROM DATA TAPE SO SIMPLY":
    PRINT "TYPE IN THE DATE AND THEN YOUR NEW LETTER":
    PRINT
220 INPUT "TYPE IN DATE FOR LETTER (00/00/00)";T:
    IF T = "9876" AND P9 = 99
     THEN
      220:
     ELSE
      IF T = "9876" AND P9 < > 99
       THEN
        200:
       ELSE
        GOSUB 230:
        GOTO 250
230 FOR K = 1 TO 1000:
     NEXT :
    CLS :
    PRINT @965,"YOUR LETTERHEAD, NAME AND P.S. WILL APPEAR LIKE THIS
    .":
    FOR HT = 1 TO 1000:
     NEXT :
    GU = 0:
    KO = 0:
    W = 0:
    CLS :
    PRINT TAB(35)Q:
    PRINT TAB(35)R:
    PRINT TAB(35)T:
    IF GU = 99
     THEN
      X = 0:
      PRINT @X,:
      GOSUB 380:
      GOTO 240:
     ELSE
      PRINT
240 PRINT TAB(35)"SINCERELY,":
    PRINT TAB(35) STRING$(20,95):
    PRINT TAB(35)I:
    PRINT "P.S. CALL ME AT ";F:
    PRINT STRING$(63,34):
    FOR HT = 1 TO 1000:
     NEXT :
    P9 = 99:
    RETURN
250 CLS :
    GOSUB 370
260 IF KO < 0
```

Program continued

```
     THEN
      KO = Ø:
      W = Ø:
     ELSE
      PRINT @832,"LINE # ";W + 1; TAB(62) CHR$(92):
      INPUT W$(W):
      IF W$(W) = "9876"
       THEN
        W$(W) = "":
        W = W - 1:
        KO = KO - 1:
        GOTO 26Ø:
       ELSE
        IF W$(W) = "9999"
         THEN
          28Ø:
         ELSE
          W = W + 1:
          KO = KO + 1:
          GOSUB 37Ø:
          IF KO = 1Ø
           THEN
            CLS :
            GOTO 28Ø
27Ø IF KO = 9 FOR KL = 1 TO 1Ø:
     FOR OY = 1 TO 5Ø:
      NEXT :
     PRINT @896," ##### ONE MORE LINE #####"; CHR$(229):
     FOR OI = 1 TO 3Ø:
      NEXT :
     PRINT @896, CHR$(222);:
     NEXT :
    GOTO 26Ø:
     ELSE
      26Ø
28Ø W$(W) = "":
    GU = 99:
    GOTO 2Ø
29Ø CLS :
    PRINT @852,"SIGNAL TO SAVE ADDRESS":
    PRINT TAB(23)"REWIND DATA TAPE":
    PRINT TAB(22)"PRESS  PLAY/RECORD":
    PRINT TAB(26):
    INPUT "HIT 12345";L:
    IF L = 12345
     THEN
      3ØØ:
     ELSE
      2Ø
3ØØ CLS :
    PRINT @464,"SAVING ALL INFORMATION";:
    PRINT # - 1,KO,P9,GU,Q,R,T,F,I:
    GOTO 2Ø
31Ø CLS :
    PRINT @852,"SIGNAL TO RETRIEVE ADDRESS":
    PRINT TAB(25)"REWIND DATA TAPE":
    PRINT TAB(27)"PRESS PLAY":
    PRINT TAB(27):
    INPUT "HIT 54321";L:
    IF L < > 54321
     THEN
      2Ø:
     ELSE
      CLS :
      PRINT @852,"RETRIEVING ALL INFORMATION";:
      INPUT # - 1,KO,P9,GU,Q,R,T,F,I:
      GOTO 2Ø
32Ø CLS :
    LPRINT STRING$(64,45):
    FOR LP = 1 TO (32 - KO) / 2:
```

```
     LPRINT CHR$(138):
     NEXT :
    PRINT TAB(35)Q:
    LPRINT TAB(35)Q:
    PRINT TAB(35)R:
    LPRINT TAB(35)R:
    PRINT TAB(35)T:
    LPRINT TAB(35)T:
    FOR LP = 1 TO 4:
     LPRINT CHR$(138):
     NEXT :
    PRINT B:
    LPRINT B:
    PRINT C:
    LPRINT C:
    PRINT D:
    LPRINT D:
    FOR LP = 1 TO 2
330  LPRINT CHR$(138):
     NEXT :
    FOR G = 1 TO LEN(B):
     IF MID$(B,G,1) = CHR$(32)
      THEN
       340:
      ELSE
       NEXT
340 PRINT @448,"DEAR ";:
    LPRINT "DEAR ";:
    FOR K = 0 TO G - 1:
     PRINT @453, LEFT$(B,K);:
     A9 = LEFT$(B,K):
     NEXT :
    LPRINT A9;:
    PRINT ",":
    LPRINT ",":
    FOR LP = 1 TO 4:
     LPRINT CHR$(138):
     NEXT :
    FOR G = 0 TO KO:
     PRINT W$(G):
     LPRINT W$(G):
     NEXT :
    FOR LP = 1 TO 5:
     LPRINT CHR$(138):
     NEXT :
    PRINT TAB(35)"SINCERELY,"
350 LPRINT TAB(35)"SINCERELY,":
    FOR G = 1 TO 2:
     LPRINT CHR$(138):
     NEXT :
    PRINT TAB(35) STRING$(20,95):
    LPRINT TAB(35) STRING$(20,95):
    PRINT TAB(35)I:
    LPRINT TAB(35)I:
    FOR LP = 1 TO 4:
     LPRINT CHR$(138):
     NEXT :
    PRINT "P.S. CALL ME AT ";F:
    LPRINT "P.S. CALL ME AT ";F
360 FOR LP = 1 TO (32 - KO) / 2:
     LPRINT CHR$(138):
     NEXT :
    LPRINT STRING$(64,45):
    RETURN
370 PRINT @780,"9999 TO END  :  ERROR - HIT 9876":
    PRINT @0,:
    FOR G = 0 TO KO - 1:
     PRINT W$(G):
     NEXT :
    RETURN
```

Program continued

```
  380 FOR G = 0 TO KO:
       PRINT W$(G):
       NEXT :
      RETURN
  390 CLS :
      PRINT @960,"PRINT ALL DATA STATEMENTS EXACTLY LIKE THIS.":
      PRINT "ANY STATEMENTS THAT HAVE COMMAS MUST HAVE QUOTE MARKS":
      PRINT
  400 PRINT "1000 DATA DR. JON MARK O'CONNOR,56 EUSTIS PARKWAY,";
      CHR$(34);"WATERVILLE, MAINE 04901"; CHR$(34);",";"1-207-555-1212
      ":
      PRINT :
      PRINT TAB(20):
      INPUT "HIT ENTER";L:
      PRINT @896, CHR$(222):
      LIST 1000 - 29999
 1000 DATA WAYNE GREEN,80 MICROCOMPUTING,"PETERBOROUGH, NEW HAMPSHIRE
      03458",1-603-555-1212
 1001 DATA DAVID SYLVER,EDITORIAL STAFF,80 MICROCOMPUTING,
30000 DATA *,*,*,*
```

Inventory

by Michael A. Rigsby

Often the proprietor of a small business works late at night and all weekend attempting to keep track of inventory and daily transactions. A computer system can act as a cash register, calculate taxes, print receipts, tally transactions, maintain inventory records, and request reorders. You can keep track of 800 items with this system, and the addition of a disk drive would boost the capacity to over 3000 types of merchandise.

System Operation

Load Inventory. Next, type RUN and press ENTER.

WHAT IS TODAY'S DATE?_

will be displayed on the screen. Enter the date desired, without commas; press ENTER.

TO OPERATE, TYPE '1'.
TO USE INVENTORY, TYPE '2'.
?_

For a cash register type operation, type 1; for other functions, type 2. Assume that a 2 was typed and entered.

MENU
TYPE THE NUMBER REPRESENTING YOUR CHOICE

1 ALTERING STOCK VALUES
2 ITEMS IN ALARM
3 VIEWING STOCK VALUES
4 RECORDING INVENTORY DATA
5 READING INVENTORY FROM TAPE
6 MODIFYING TAX PERCENTAGE
7 EXAMINATION OF DAY'S RECEIPTS
8 PRINTING OF INVENTORY VALUES AND ALARMS
?_

A 1 enables you to enter stock values and alarm limits. A 2 causes the computer to search through the inventory for items in alarm. Any items in alarm will be displayed on the monitor; they will be printed by the printer if you answer YES to the question, DO YOU WISH HARD COPY OF THE ALARMS?. The machine takes about 30 seconds to perform this search. A 3 enables the operator to view inventory values and alarm values on the monitor, one at a time. A 4 records the values within the computer onto cassette tape (Inventory Data I or Inventory Data II). This requires about 12 minutes for completion. 5 moves inventory data from

the appropriate tape (Inventory Data I or Inventory Data II) into the computer. This also requires about 12 minutes. A 6 lets you examine or change the sales tax rate. This number needs to be entered only once, as it will become part of the information stored and loaded from the inventory data tapes. A 7 causes a display to be produced on the monitor similar to the one below.

TOTAL INPUT FOR THE DAY IS 3.09
TOTAL CHANGE FOR THE DAY IS 0
TOTAL CASH IN TODAY WAS 3.09
TOTAL CHECKS IN TODAY EQUALED 0
TOTAL CREDIT SALES TODAY AMOUNTED TO 0
TOTAL TAXES FOR TODAY EQUALED .09
HIT ENTER TO RETURN TO START MODE
?_

An 8 causes the printer to produce a copy of all stock numbers and the inventory available as well as the alarm values which have been set.

Returning to the start:

TO OPERATE, TYPE '1'.
TO USE INVENTORY, TYPE '2'.
?_

Typing a 1 and pressing ENTER evokes a functional set of questions.

WHAT IS STOCK NUMBER?_ (Answer and ENTER)
HOW MANY ITEMS ARE BEING PURCHASED?_ (Answer and ENTER)
WHAT IS THE PRICE PER ITEM?_ (Answer and ENTER)

The stock number, quantity, price, and subtotal for that item will be displayed on the monitor and printed.

ANOTHER ITEM?_ (Answer Y or YES or NO and ENTER)

If the answer is yes, the previous questions will be repeated, if no, new responses appear. The subtotal, taxes, and total appear on the monitor and at the printer.

HOW MUCH MONEY WAS TENDERED? (Answer and ENTER)
WAS IT CASH (1); CHECK (2); OR CREDIT (3)?_
(Answer "1", or "2", or "3" and ENTER)
THE CHANGE BACK IS (answer)
HIT ENTER TO CONTINUE?_

Upon depressing ENTER, the monitor will clear and WHAT IS THE STOCK NUMBER? will occur again. To get back to the menu, type the word MODE after WHAT IS THE STOCK NUMBER? and the program will return to start. Table 1 shows a sample sales receipt.

As with any automated system, preparations are required to initiate the system after which it will be simple to maintain. You must assign a stock number to each item in the inventory—any number between two and 804. A card file should be made, one card for each number. On the card should be the vendor's name and address along with any information pertinent to ordering or maintaining that inventory item. Each item within the inven-

# 34 (2) @ 3.45 = 6.9
# 35 (1) @ 5.67 = 5.67
# 2 (1) @ .45 = .45
# 567 (3) @ 3.89 = 11.67
# 45 (3) @ .99 = 2.97
# 456 (1) @ .23 = .23
SUBTOTAL = 27.89
TAX = .84
TOTAL = 28.73
TENDERED = 30
CASH
THE CHANGE BACK IS 1.27
AUGUST 21 1980

Table 1. *Sample sales receipt*

tory must be counted for transference to the computer system. Everything should be tagged with a stock number and the retail price. After completion, this data must be entered. Once the program is established, it will never be necessary to take inventory again. The machine will maintain accurate records. At predetermined levels (you select the level for each item) an alarm will be available to indicate that it is time to reorder. A provision is included to store vital information about each transaction on magnetic tape as the activity occurs. This insures that no transaction need be lost due to power failure.

Within the keyboard unit (central processor) are memory circuits (RAM—random access memory). The memory may be filled with instructions (instructions are called programs; programs are called software) and data. Data is information (numbers and words) which the operator wishes to manipulate. Random access memory (RAM) functions only while the computer is on. If the computer is turned off or if power fails—for as little as one second— the machine "forgets" everything it had in RAM.

The system requires four cassette tapes to operate. The first is the Inventory program. The Recover program may also be recorded on the same cassette. Two tapes are reserved for inventory data; these hold the present status of the inventory. Two data tapes are necessary—one for the morning and one for the evening—to assure preservation of the inventory list. Suppose a power failure occurs in the evening while the day's inventory is being transferred to tape. That tape and the computer memory would contain meaningless data. The Recover Data tape (transactions of one day) and the morning inventory tape (the correct inventory list before the present business day) could be used to rebuild the current inventory

list and transfer it to tape (at which point the new tape would become the next day's "morning" tape). The fourth tape is Recover Data, used if the computer "forgets," due to power loss.

On a typical day, the machine must be turned on and Inventory loaded. Next, the Inventory Data tape (the one with the more current inventory values) must be read into the computer. Before operating the system, the Recover Data tape must be rewound and placed into the recorder with the recorder set up in the record mode. At the end of the day pull the microphone plug out of the recorder and allow about five seconds of nothing to be recorded on the tape. This is done so that the Recover Program can find the end of data if necessary. Insert the Inventory Data tape that is not most current (do not use the same one used in the morning), and record the inventory data onto the cassette. Label the Inventory Data tapes in such a manner as to know which is most current. It will be used in the morning.

STOCK NUMBER	QUANTITY
34	2
35	1
2	1
567	3
45	3
456	1
0	0
0	0
0	0
0	0
654	3
46	4
0	0
0	0
0	0

Table 2. *Recovered data*

Trouble?

Incorrect entries may be modified easily. If an incorrect stock number, quantity, or price is entered, complete the information requested until ANOTHER ITEM?__ is displayed. Carefully observing the incorrect line, repeat the stock number as previously typed; repeat the quantity, preceding it with a minus sign; repeat the price as previously typed. This will negate an incorrect line and the system will be ready to operate normally.

If the power fails, record about five seconds of nothing on the Recover Data tape as described previously. Remove that tape. Load the Recover program. Return the Recover Data cassette to the recorder and place the recorder in the play mode. Recover causes the printer to list the stock number and quantity of each item purchased until the power failure. It may or may not contain the transaction in progress when the power failed. Load Inventory and then load the inventory data tape from the morning. The printed list prepared by Recover must be entered manually into the computer. Table 2 shows a sample list of recovered data.

Obviously this will take time, possibly more than can be spared in the middle of a business day. Another option would be to remove the Recover Data tape and load Inventory. Modify the tax percentage in the inventory program (put in the sales tax rate); reload the Recover Data cassette (placing the recorder in record mode) and run the program. Doing this will take two or three minutes; the Recover program can be used at the end of the day to get the inventory straight.

Conclusion

When operating Inventory, a time delay of about five seconds occurs after five stock numbers are entered or when ANOTHER ITEM?__ is answered with NO. Stock numbers and quantities purchased are being recorded on Recover Data during this time. The 12 minutes consumed transferring the inventory data to and from tape are directly proportional to the number of items in the inventory list; if 1400 items were used, the time would be about 21 minutes. If you want to program to handle more than 800 items, additional memory must be acquired. A few program changes are necessary to handle additional items and they are listed below:

```
5 DIM INV (804,2)
410 IF S=805 THEN 490
512 IF I>804 THEN 515
606 IF D>804 THEN 1900
1000 IF B>804 THEN 1100
1325 IF A=804 THEN 1350
1425 IF A=804 THEN 1450
```

In all lines listed except 410 the number 804 appears. 804 must be changed to the new limit, the new limit being a number which is divisible by six with no remainder. The 805 in line 410 must equal the new limit plus one.

The program, written in BASIC, uses a two-dimensional array to store the inventory information. Most program lines contain single statements and as such are not overly difficult to follow. Start with a small system, learn how it works and how to change things; then it will be easy to assess the feasibility of further expansion.

Program Listing 1. *Inventory*

```
  5 DIM INV(8Ø4,2)
  6 CLS
  7 INPUT "WHAT IS TODAY'S DATE";Q$
 1Ø CLS
 15 PRINT "TO OPERATE, TYPE '1'."
 2Ø PRINT "TO USE INVENTORY, TYPE '2'."
 25 INPUT "";A
 3Ø IF A = 1
     THEN
      4Ø
 31 IF A = 2
     THEN
      5Ø
 35 GOTO 1Ø
 4Ø CLS
 45 INPUT "WHAT IS STOCK NUMBER";B
 46 IF INV(1,1) = Ø
     THEN
      17ØØ
 47 GOTO 1ØØØ
 5Ø CLS
 51 PRINT "           MENU"
 52 PRINT " "
 54 PRINT " "
 55 PRINT "TYPE THE NUMBER REPRESENTING YOUR CHOICE."
 57 PRINT
 6Ø PRINT "1     ALTERING STOCK VALUES"
 62 PRINT "2     ITEMS IN ALARM"
 65 PRINT "3     VIEWING STOCK VALUES"
 66 GOTO 55Ø
 7Ø GOTO 1Ø
2Ø5 O PRINT "THE PRESENT SALES TAX RATE IS ";INV(1,1)
4ØØ CLS
4Ø1 GOTO 4ØØØ
4Ø2 S = Ø
4Ø5 S = S + 1
41Ø IF S = 8Ø5
     THEN
      49Ø
415 A = INV(S,1)
42Ø B = INV(S,2)
425 IF A < B
     THEN
      43Ø
426 GOTO 4Ø5
43Ø PRINT "STOCK NUMBER";S;"QUANTITY";A;"ALARM VALUE";B
432 IF VV = 1
     THEN
      LPRINT "STOCK NUMBER";S;"  ";A;"   ";B
435 GOTO 4Ø5
49Ø INPUT "TYPE ENTER WHEN YOU WISH TO CONTINUE";Z$
495 GOTO 1Ø
5ØØ CLS
5Ø5 PRINT "YOU ARE IN THE STOCK ALTERING MODE."
51Ø INPUT "WHAT IS THE STOCK NUMBER YOU WISH TO EXAMINE";I
512 IF I > 8Ø4
     THEN
      515
513 IF I < 2
     THEN
      515
514 GOTO 52Ø
515 PRINT "YOU HAVE CHOSEN AN INVALID NUMBER, TRY AGAIN"
516 GOTO 51Ø
52Ø PRINT "HOW MANY OF THE ITEMS ARE THERE?"
525 INPUT INV(I,1)
53Ø PRINT "WHAT IS THE PRESET LIMIT?"
```

```
535 INPUT INV(I,2)
537 INPUT "DO YOU WISH TO ALTER ANOTHER NUMBER";B$
540 IF B$ = "NO"
     THEN
      10
545 GOTO 500
547 GOTO 50
550 PRINT "4      RECORDING INVENTORY DATA"
552 PRINT "5      READING INVENTORY FROM TAPE"
553 PRINT "6      MODIFYING TAX PERCENTAGE"
554 PRINT "7      EXAMINATION OF DAY'S RECEIPTS"
555 GOTO 5000
556 GOTO 1500
557 GOTO 1200
560 GOTO 10
600 CLS
605 INPUT "WHAT NUMBER DO YOU WISH TO VIEW";D
606 IF D > 804
     THEN
      1900
610 PRINT "THE STOCK NUMBER IS";D
615 PRINT "PRESENTLY THERE ARE";INV(D,1)
620 PRINT "THE ALARM VALUE IS";INV(D,2)
625 INPUT "DO YOU WISH TO VIEW ANOTHER NUMBER";D$
630 IF D$ = "YES"
     THEN
      600
640 GOTO 10
1000 IF B > 804
     THEN
      1100
1002 IF B < 2
     THEN
      1100
1005 INPUT "HOW MANY ITEMS ARE BEING PURCHASED";C
1010 D = INV(B,1)
1015 D = D - C
1020 INV(B,1) = D
1025 INPUT "WHAT IS THE PRICE PER ITEM";E
1030 F = C * E
1035 PRINT "# ";B;"  QUANTITY ";C;"  PRICE ";E;"  TOTAL   ";F
1037 LPRINT "#";B;"(";C;")  @";E;"=";F
1038 GOTO 7000
1040 G = F + G
1045 INPUT "ANOTHER ITEM";A$
1050 IF A$ = "Y"
     THEN
      45
1052 IF A$ = "YES"
     THEN
      45
1053 GOTO 7030
1054 PRINT "      SUBTOTAL=";G
1055 LPRINT "      SUBTOTAL=";G
1056 H = INV(1,1)
1057 K = G * H
1058 K = INT(K * 100 + .5) / 100
1059 GOTO 1800
1060 INPUT "HIT ENTER TO CONTINUE";A$
1061 G = 0
1065 IF A$ = "MODE"
     THEN
      10
1068 CLS
1070 GOTO 45
1100 PRINT "INVALID STOCK NUMBER, PLEASE TRY AGAIN"
1105 GOTO 45
1200 INPUT "DO YOU WISH TO RECORD THE INVENTORY DATA";H$
1205 IF H$ = "YES"
     THEN
```

Program continued

```
      1300
1210 INPUT "DO YOU WISH TO READ INVENTORY FROM TAPE";H$
1215 IF H$ = "YES"
      THEN
      1400
1220 GOTO 10
1300 CLS
1305 INPUT "PREPARE THE TAPE RECORDER, HIT ENTER WHEN READY";A
1310 A = - 6
1315 A = A + 6
1320 PRINT # - 1,INV(A,1),INV(A,2),INV((A + 1),1),INV((A + 1),2),INV(
     (A + 2),1),INV((A + 2),2),INV((A + 3),1),INV((A + 3),2),INV((A
     + 4),1),INV((A + 4),2),INV((A + 5),1),INV((A + 5),2)
1325 IF A = 804
      THEN
      1350
1330 GOTO 1315
1350 PRINT
1355 GOTO 10
1360 INPUT "HIT ENTER TO RETURN TO START";A
1365 GOTO 10
1400 CLS
1405 INPUT "PREPARE THE TAPE RECORDER, HIT ENTER WHEN READY";A
1410 A = - 6
1415 A = A + 6
1420 INPUT # - 1,INV(A,1),INV(A,2),INV((A + 1),1),INV((A + 1),2),INV(
     (A + 2),1),INV((A + 2),2),INV((A + 3),1),INV((A + 3),2),INV((A
     + 4),1),INV((A + 4),2),INV((A + 5),1),INV((A + 5),2)
1425 IF A = 804
      THEN
      1450
1430 GOTO 1415
1450 PRINT
1455 PRINT "FINISHED INPUTTING"
1460 INPUT "HIT ENTER TO RETURN TO START";A
1465 GOTO 10
1500 ON A GOTO 500,400,600,1300,1400,2000,2100,6000
1530 PRINT "YOUR CHOICE WAS INVALID, HIT ENTER TO TRY AGAIN."
1535 INPUT A
1536 GOTO 10
1700 CLS
1705 PRINT "THE TAX VALUE IS 0, THE PROGRAM REQUIRES SOME TAX FOR OPE
     RATION"
1710 PRINT "HIT ENTER AND GO TO TAX TABLE TO ENTER TAX VALUE."
1715 INPUT A
1720 GOTO 10
1800 PRINT "           TAX=";K
1801 LPRINT "           TAX=";K
1802 GOTO 2500
1805 L = G + K
1810 PRINT "         TOTAL=";L
1811 LPRINT "         TOTAL=";L
1815 INPUT "HOW MUCH MONEY WAS TENDERED";Z1
1816 LPRINT "      TENDERED=";Z1
1820 INPUT "WAS IT CASH (1); CHECK (2); OR CREDIT (3)";Z2
1821 GOTO 3000
1822 ON Z2 GOTO 2200,2300,2400
1825 GOTO 2160
1830 Z3 = 0
1831 Z3 = Z1 - L
1832 IF Z3 < .01
      THEN
      2600
1833 PRINT "THE CHANGE BACK IS ";Z3
1834 LPRINT " THE CHANGE BACK IS";Z3
1835 LPRINT Q$
1837 LPRINT
1838 LPRINT
1840 GOTO 2150
1900 PRINT "INVALID CHOICE, TRY AGAIN"
```

```
1905 GOTO 605
2000 CLS
2005 INPUT "DO YOU WISH TO VIEW THE PRESENT SALES TAX";A$
2010 IF A$ = "YES"
      THEN
       2050
2015 PRINT "WHAT IS THE SALES TAX YOU WISH TO INPUT?"
2020 INPUT INV(1,1)
2025 PRINT "HIT INPUT TO RETURN TO START"
2030 INPUT A
2035 GOTO 10
2050 PRINT "THE PRESENT SALES TAX IS ";INV(1,1)
2060 GOTO 2025
2100 CLS
2110 PRINT "TOTAL INPUT FOR THE DAY IS ";Z4
2115 PRINT "TOTAL CHANGE BACK FOR THE DAY IS ";Z6
2120 PRINT "TOTAL CASH IN TODAY WAS ";Z7
2125 PRINT "TOTAL CHECKS IN TODAY EQUALED ";Z8
2130 PRINT "TOTAL CREDIT SALES TODAY AMOUNTED TO ";Z9
2135 PRINT "TOTAL TAXES FOR TODAY EQUALED ";K1
2137 M9 = Z4 - Z6 - K1
2138 PRINT "THE TOTAL INCOME FOR THE DAY, LESS TAXES IS ";M9
2140 PRINT "HIT ENTER TO RETURN TO START MODE"
2145 INPUT A
2147 GOTO 10
2150 Z4 = Z1 + Z4
2155 Z6 = Z3 + Z6
2157 GOTO 1060
2160 PRINT "INVALID CHOICE"
2165 GOTO 1820
2200 Z7 = Z1 + Z7
2210 GOTO 1830
2300 Z8 = Z1 + Z8
2310 GOTO 1830
2400 Z9 = Z1 + Z9
2410 GOTO 1830
2500 K1 = K + K1
2510 GOTO 1805
2600 Z3 = 0
2610 GOTO 1833
3000 ON Z2 GOTO 3050,3055,3060
3015 GOTO 1822
3050 LPRINT "         CASH"
3052 GOTO 1822
3055 LPRINT "        CHECK"
3057 GOTO 1822
3060 LPRINT "        CHARGE"
3062 GOTO 1822
4000 INPUT "DO YOU WISH HARD COPY OF THE ALARMS";A$
4005 VV = 0
4010 IF A$ = "YES"
      THEN
       VV = 1
4020 PRINT "SEARCHING FOR ITEMS IN ALARM"
4030 GOTO 402
5000 PRINT "8      PRINTING OF INVENTORY VALUES AND ALARMS"
5005 A = 0
5010 INPUT A
5020 GOTO 1500
6000 S = 0
6010 S = S + 1
6020 LPRINT S;"   ";INV(S,1);"   ";INV(S,2)
6030 IF S = 800
      THEN
       10
6040 GOTO 6010
7000 L2 = L2 + 1
7010 ON L2 GOTO 7100,7200,7300,7400,7500
7030 PRINT # - 1,L3,L4,L5,L6,L7,L8,M4,M5,M6,M7
```

Program continued

```
7035 L3 = Ø:
     L4 = Ø:
     L5 = Ø:
     L6 = Ø:
     L7 = Ø:
     L8 = Ø:
     M4 = Ø:
     M5 = Ø:
     M6 = Ø:
     M7 = Ø
7040 L2 = Ø:
     GOTO 1Ø54
7100 L3 = B:
     L4 = C:
     GOTO 1Ø4Ø
72ØØ L5 = B:
     L6 = C:
     GOTO 1Ø4Ø
73ØØ L7 = B:
     L8 = C:
     GOTO 1Ø4Ø
74ØØ M4 = B:
     M5 = C:
     GOTO 1Ø4Ø
75ØØ M6 = B:
     M7 = C
751Ø PRINT # - 1,L3,L4,L5,L6,L7,L8,M4,M5,M6,M7
7515 L3 = Ø:
     L4 = Ø:
     L5 = Ø:
     L6 = Ø:
     L7 = Ø:
     L8 = Ø:
     M4 = Ø:
     M5 = Ø:
     M6 = Ø:
     M7 = Ø
752Ø L2 = Ø:
     GOTO 1Ø4Ø
```

Program Listing 2. *Recover*

```
  5 REM  R E C O V E R
 1Ø CLS
 2Ø INPUT "PREPARE TAPE RECORDER AND HIT ENTER WHEN READY";A
 3Ø CLS
 4Ø PRINT "STOCK NUMBER","QUANTITY"
 5Ø LPRINT "STOCK NUMBER","QUANTITY"
 6Ø INPUT # - 1,A,B,C,D,E,F,G,H,I,J
 7Ø PRINT A,B
 8Ø LPRINT A,B
 9Ø PRINT C,D
1ØØ LPRINT C,D
11Ø PRINT E,F
12Ø LPRINT E,F
13Ø PRINT G,H
14Ø LPRINT G,H
15Ø PRINT I,J
16Ø LPRINT I,J
17Ø GOTO 6Ø
```

EDUCATION

Algebra Tutor

EDUCATION

Algebra Tutor

by Anne Weiss

In September of 1980, St. Peter's High School in New Brunswick, NJ invited 20 eighth graders to take an Algebra I course. Our purpose was to offer non-traditional material to these above-average students and introduce the students to the computer—a 16K Level II TRS-80. Since I had some experience developing usable programs I decided to develop our programs.

Six programs are listed at the end of this chapter. Many REM statements and a list of variables are included for each one. Look at the beginning REM statements to find the line number to go to if you want to see scores when the program is interrupted at the end of a class period before all the students are finished.

Program Listing 1

Program Listing 1 serves many purposes. Basically it is an introduction to using the computer. The program also reviews arithmetic and order of operations.

First, the students are shown how to run a program. Once that is done, they see the directions on the screen as well as the symbols for zero (Ø) and for multiplication (*). The program then goes on to take roll. We did this because only five or six students use the computer at a given time, and we never know who will be in the group for any one use. Besides, it lends a nice personal touch. At first, they didn't know that their names were entered as DATA statements. This leads to a discussion of what computers can and cannot do.

Students who are currently using the computer are given four arithmetic questions of the form X(Y + Z) or XY + Z. The variables are randomly chosen with X ranging from .1 to 30, Y from 1 to 50, and Z from 1 to 90. If a wrong answer is entered, the student is told the correct one. This stimulates finding out what was done incorrectly. At the end of four problems, the student is given his or her score.

After everyone has a turn, all scores are shown. Students not receiving 100 percent are then given a chance to answer four new problems. An appropriate message is given at the end of each student's turn, depending on whether the score was raised, lowered, or remained the same. The program continues until each student achieves 100 percent. With only five or six in each group, this is usually done within a 40-minute period. We find that any student who gets below 100% after the second round receives

"group tutoring" from his classmates on the next turn.

We discovered that we can use this program with our arithmetic students also. A few DATA statement changes is all that it takes along with changing the number-of-students variable (N) in line 120. We have used this program both with and without calculators and are pleased with it either way.

We plan to use this program for drill and practice of other concepts. All that has to be done is to change the formula within the loop in lines 370–500 and the random variable assignment in lines 890–910.

Program Listing 2

We were quite pleased with the success of Program Listing 1. The students pestered us to let them use the computer again and didn't want to wait their group's turn. At this point we were still reviewing decimal arithmetic with the eighth graders. We decided to break them into groups of ten this time, and to let them work at their seats on a review sheet.

Program Listing 2 was written as a computerized answer-key. By now, we were very aware of the need for variety with this group of youngsters. Therefore, the format of Program Listing 2 had to be different from the last one. Music made that difference! Thanks to one of my computer students, John Dondzila, I was able to insert a small assembly-language routine in the BASIC program. The AUX cable of the cassette recorder must be plugged into an amplifier (we use the small Radio Shack one).

The music program is loaded into memory by a series of POKE statements. Line 120 takes care of reserving the 28 memory locations needed for the program. The decimal address of 32738 works with a 16K Model I. If you are using something else, find the value of the top of your memory and subtract 28. Substitute that value for AD in line 120. To run the program without the music, simply delete lines 90–160, 540, 610–622.

After working at their seats on the decimal review sheet, the students come up to enter their answers into the computer. This time, the student sees a list of his or her classmates on the screen. For each student there is a number, his or her name, and his or her score thus far. After entering his or her number, the student enters the answer to each problem. Should the student give a wrong answer, a raspberry-like buzzer is emitted and the student is told to redo the problem. When the student comes back to the computer, he or she will continue the sequence of problems. This continues until all problems are answered correctly. When that happens, a musical "cheer" sounds. And so it goes until each student gets 100 percent.

This program can easily be adapted to be an answer-key to any work sheet. The value of P (number of problems) is in line 190. The answers are in lines 730–750.

Table 1 shows the review sheet for Program Listing 2.

NAME ______________________ DATE ______________________

CLASS ______________________ TEACHER ______________________

Review of Decimal Fraction Operations

Compute these decimal problems. Place your answers in the space at the bottom of the page.

1. $.3\overline{)3.75}$	2. $.08\overline{)1.888}$	3. $.31\overline{)10.757}$
4. 3.206 ÷ .7 =	5. 11.82 − 6.13 =	
6. $13.42 × 5	7. 8.69 × .9	8. 13. −4.068
9. 200.653 −109.61	10. 37.5 − 27.2846 = 11. 4.2653 × 5 =	
12. 21.257 8.6 + 2.5806	13. 10.00001 −5.64514	
14. 1.6 1.73 1.124 .2574 + .75458	15. 100. −34.23891	

Table 1. *Work sheet for Program Listing 2*

Program Listing 3

Eventually we got around to introducing some algebra concepts. Program Listing 3 serves as drill and practice for order of operations using signed numbers.

The program starts by having someone take roll for the whole class. Since only five or six students use the program at any one time, this is a necessary chore. As with Program Listing 1, absent students are flagged by having their scores, S(I), set equal to −1 in line 230. Once a student is marked absent, he or she is not called on again for the duration of the program.

I decided to use the idea of a contest as the format for this program. The object is to see which student can get the most right out of five problems.

Students must simplify problems of the form $X + (Y + Z)$ or $-X-Y+Z$ or $-(X-Y)+Z$ or $-(X+Y+Z)$ or $(-X+Y)+(-Z)$ where X, Y, and Z are integers or decimals between -20 and 20, but not zero.

As the program is listed here, there are five turns per student. To change that number, adjust the value of T in line 110. The student is told the correct answer whenever he or she gives a wrong reply. After each student has had five turns, the name of the student with the highest score appears on the screen.

Program Listing 4

By now, the students were solving linear equations of the form $Ax+By=C$ where A, B and C are decimals or whole numbers. Program Listing 4 offered the students a chance to solve such equations in any one of four levels of difficulty. Level 1 generates equations of the form $X+12=5$. Level 2 gives slightly harder ones, such as $X+22.9=47.1$. Then comes level 3 with $5X-3=-8$. Finally, level 4 offers equations of the form $-3X+17.4=94.1$. The students choose their own level of difficulty, which means they usually try the hard ones first.

Students entering an incorrect answer are given the correct one, so that they may check their work. The program is written to give every student five turns. To change that value, adjust T in line 110.

This program also starts out by having someone take roll. It ends after five turns with screen display of the students' scores. There is a provision here for the program to be repeated (line 440). If it is, the students have a chance to change the level of difficulty before getting more equations to solve.

Program Listing 5

It was time to change our program format. We were now aware that the eighth graders enjoyed competition. They not only were pestering us to let them compete against each other, but were also bragging about being able to take on the top level Algebra I classes of ninth graders. Time would not allow the luxury of having 55 more students work individually with the computer. I therefore wrote the next program to include teams as well as individuals. Instead of having student names as part of the program data, the kids are asked to enter their names.

To heighten the element of competition, a timer was added to the programs. Program Listing 5 allows up to two minutes to answer problems of the form $.05X+.04(500-X)=22$.

There are 24 equations with their answers stored as data. Each student or team (up to six) gets four turns at solving randomly chosen equations. Once an equation has been correctly solved, it no longer can be given as a

problem. This is accomplished by setting correctly answered equations equal to the null string, E$(X) = "", in line 460. One of the 24 equations is randomly chosen in line 340, using X=RND(24). If that equation is the null string, X is increased by 1 until a non-null one is found (lines 350–360). To keep X within bounds, a wrap-around is provided in line 350 (IF X=25 THEN X=1).

The timed input didn't seem like much of a challenge until the students were turned loose on the program. I never knew kids could push so many wrong keys. The timer routine was then rewritten to include only what we want, and to exclude everything else.

The timer-counter C is set at 5000 for two minutes time. H$ holds the pieces of the student's answer until ENTER is pushed. The routine starts by clearing both in line 380. It then scans the keyboard using INKEY$, increments the counter, checks if time is left, and goes back to the keyboard scan. The loop is broken either when time runs out (lines 390 and 410) or ENTER is pushed (line 420). Only numbers or the decimal point are accepted, printed on the screen, and stored in H$ (lines 400 and 440). The back-arrow is accepted only if there is something in H$. In that case, the previous character is deleted from both the screen and H$ (line 430). When ENTER is pressed, A is set to the final answer (lines 420 and 450). If time runs out, the student is told that he or she took too long to answer (lines 390, 410, and 650). The next student then gets called upon.

Lines 500–520 and line 120 allow for a flashing message to be displayed at the bottom of the screen. It's the old "Press ENTER to continue" with a new set of clothes.

Program Listing 6

The final program (Program Listing 6) covers five different work sheets, each containing 30 different problems of a given type. Only two are listed in the program to save space. Tables 2 and 3 show sample work sheets. Up to six individuals or teams can use the program at one time.

After finding out the names of the students and which work sheet they are using, the program proceeds to assign different problems to each student. Lines 310–380 read the correct answers, randomly pick five problems for each student or team, and display those problem numbers on the screen. F(X) serves as a flag for which problems have (1) or have not (0) been assigned, so that no two students get the same problem. H(I,J) serves as a holder for the Ith student's problem number. For example, H(3,1) . . . H(3,5) are the numbers of the five problems assigned to student number three.

Once the problems have been assigned, the students work out their respective equations from the given work sheet. I usually give them five or

six minutes before asking them to enter some answers. To keep things moving, the program is designed to allow students to skip any unanswered problems for the time being by just pushing ENTER (lines 450, 480, 490). In that case, they are told to try that one later. This way, we don't have to wait for everyone to finish before answers are checked. Also, any incorrectly answered problems must be redone and entered on the student's

Do not write on these sheets. Use scrap paper to solve for X.

1.	.8X = 5.68	2.	X + .7 = 5.4
3.	.04X = 100	4.	.5X = 20
5.	1.04X = 52	6.	.7X = 2.1
7.	1.25X = 625	8.	.3X = .6
9.	.09X = 7.2	10.	6X = .12
11.	X + .5 = 3.6	12.	.3X + .2 = .8
13.	X + 2 = 4.7	14.	.5X + .8 = 3.3
15.	X + 3.54 = 5.8	16.	2.4X + .15 = .87
17.	X − .4 = 6.25	18.	3.2X − .3 = 6.1
19.	X − 3 = .27	20.	.6X − .8 = 4
21.	X − .02 = 2.5	22.	.2X − .7 = .78
23.	.8 = X + .3	24.	.04X + 8 = 10.4
25.	12 = X − 1.4	26.	1.02X + 4.8 = 55.8
27.	X − 2 = 8.5	28.	2.5X − .5 = 24.5
29.	X + 3.6 = 9	30.	.24 + 8X = .64

Table 2. *Work sheet number 1*

Do not write on these sheets. Use scrap paper to solve for X.

1.	.3X + 1.2X = 4.5	2.	.4X + .2X = .24 + .3X
3.	3.7X + 2.1X = 8.7	4.	X + .4 = .6X + 2
5.	4.2X − 3.6X = 9.6	6.	1.8X + .5X − .48 = .7X
7.	X + .04X = 104	8.	.05X + 1.8 = .25X − .2
9.	.92X − 124 = .3X	10.	1.2X + .05X = .15X + 5.5
11.	.8X − 6 = .2X	12.	.04(X − 5) = 4
13.	X + .05X = 525	14.	.3(X − 2) = .2X
15.	1.2X − .08 = .8X	16.	.02(X + 6) = .04X
17.	40 − 3.5X = .5X	18.	.06(X − 5) = .05(X − 4)
19.	2.6X − .8 = .6X	20.	.25(X + 60) = 6 + X
21.	.6X + .3 = .3X + .9	22.	.75(X + 6) = 21
23.	X + 3.5 = .7X − .1	24.	.06X = .08(X − 50)
25.	.06X − .25 = .03X + .35	26.	.04X + .05(500 − X) = 23
27.	X + .05X + .02X = 321	28.	.03X + .02(800 − X) = 19
29.	2X + 1.08X − 30.6 = .02X	30.	.06X + .04(1500 − X) = 72

Table 3. *Work sheet number 2*

next turn. C(X) keeps track of which problems have been correctly answered.

The program continues until each student (or team) gets 100 percent. Here again, we have students helping each other after the second round of inputs. That really works well here, because the "tutors" are working on different problems than they previously solved.

Future Programs

We can adapt these programs to any group by just changing a number here or there and a few DATA statements or formula generators. The formats are varied enough so that we can keep the students from being bored by simply rotating their use.

Our next project will be fixing the programs up with some really simple questions and letting the faculty and administration use them. We must help dispel the fear of computers that can turn an intelligent human into a shaking bowl of gelatin!

Program Listing 1. *Order of operations and arithmetic*

```
10 :
   '    *    *    *    *    *    *    *    *    *    *    *    *    *    *
20 :
   '    *    *              PROGRAM  ONE                        *    *
30 :
   '    *    *     ORDER OF OPERATIONS AND ARITHMETIC           *    *
40 :
   '    *    * GOTO 590 TO DISPLAY SCORES AFTER A BREAK         *    *
50 :
   '    *    *  PROGRAM RUNS UNTIL EVERYONE GETS 100%           *    *
60 :
   '    *    *              BY ANNE WEISS                       *    *
70 :
   '    *    * ST PETER'S H.S., NEW BRUNSWICK, NJ 08901         *    *
80 :
   '    *    *    *    *    *    *    *    *    *    *    *    *    *    *
90 :
   '
100 REM  INITIALIZATION AND DIRECTIONS
110 :
   '
120 CLS :
    N = 20:
    DIM S(N):
    T = 4
130 PRINT "H E L L O !"
140 PRINT :
    PRINT :
    FOR I = 1 TO N:
     S(I) = - 2:
     NEXT
150 PRINT "LET'S SEE WHAT YOU REMEMBER ABOUT ARITHMETIC"
160 PRINT "I WILL ASK YOU SOME SIMPLE QUESTIONS."
170 PRINT "TYPE IN YOUR ANSWER AND THEN PUSH THE WHITE ENTER KEY"
180 PRINT :
    PRINT "I USE THE SYMBOL  0  FOR THE NUMBER 'ZERO'
190 PRINT "I USE THE SYMBOL  *  TO STAND FOR MULTIPLICATION"
200 PRINT "YOU WILL EACH ANSWER";T;"QUESTIONS"
210 PRINT
220 INPUT "ARE YOU READY TO BEGIN ";Q$
230 IF LEFT$(Q$,1) = "N" PRINT "NOW ";:
    GOTO 220
240 RESTORE :
    F = 0:
    CLS
250 :
   '
260 REM  MAIN LOOP...LINES 280 - 570
270 :
   '
280 FOR I = 1 TO N:
     READ N$
290  IF S(I) = T OR S(I) = - 1
      THEN
       570 :
       REM  SKIP ABSENT OR FINISHED STUDENTS
300  IF S(I) > = 0
      THEN
       330 :
       REM  TAKE ROLL ONLY ONCE
310  PRINT :
     PRINT "IS ";N$;:
     INPUT " HERE ";Q$:
     S(I) = 0
320  IF LEFT$(Q$,1) = "N"
      THEN
       S(I) = - 1:
```

```
      GOTO 570 :
      REM  FLAG ABSENT STUDENTS
330  CLS :
     C = 0
340  PRINT "AS OF NOW ";N$;" YOUR SCORE IS"; INT((100 * S(I)
     / T) + .5);"%"
350  PRINT "SEE IF YOU CAN BRING IT UP TO 100%"
360  PRINT :
     PRINT
370  FOR J = 1 TO T:
      PRINT
380   A = RND(2):
      ON A GOSUB 890 ,900 :
      REM  INTEGERS OR DECIMALS
390   D = RND(2):
      REM  CHOOSE TYPE OF PROBLEM
400   IF D = 1
       THEN
        460
410   PRINT X;"*(";Y;"+";Z;") = ";:
      INPUT R
420   C1 = X * (Y + Z):
      C1 = .01 * INT(100 * C1)
430   IF ABS(R - C1) < .0001 GOSUB 720 :
      GOTO 500
440   GOSUB 810 :
      PRINT "THE CORRECT ANSWER IS";X * (Y + Z)
450   GOTO 500
460   PRINT X;"*";Y;"+";Z;" = ";:
      INPUT R
470   C1 = X * Y + Z:
      C1 = .01 * INT(100 * C1)
480   IF ABS(R - C1) < .009 GOSUB 720 :
      GOTO 500
490   GOSUB 810 :
      PRINT "THE CORRECT ANSWER IS";X * Y + Z
500   NEXT J:
     H = S(I):
     S(I) = C:
     PRINT
510  S = INT((100 * S(I) / T) + .5)
520  IF C > H PRINT "VERY GOOD ";N$;"!  YOU HAVE INCREASED YOUR SCOR
     E TO";S;"%"
530  IF C = H PRINT "YOUR SCORE REMAINS AT";S;"%"
540  IF C < H PRINT "TOO BAD ";N$;".  YOUR SCORE HAS FALLEN TO";S;"%
     "
550  PRINT :
     PRINT :
     INPUT "PUSH ENTER TO CONTINUE ";Q$
560  IF S(I) < T
      THEN
       F = F + 1:
       REM  COUNT THOSE NOT GETTING 100% YET
570  NEXT I
580 :
    '
590 REM  PRINT SCOREBOARD
600 :
    '
610 CLS :
    RESTORE :
    FOR I = 1 TO N:
     READ N$:
     IF S(I) = - 1
      THEN
       630
620  PRINT N$, INT((100 * S(I) / T) + .5);"%",
630  NEXT I:
    PRINT :
    INPUT "PUSH ENTER TO CONTINUE ";Q$
```

Program continued

```
640 IF F > 0
     THEN
      240 :
      REM  KEEP GOING UNTIL EVERYONE GETS 100%
650 CLS :
    PRINT "CONGRATULATIONS!"
660 PRINT :
    PRINT "YOU HAVE EACH RECEIVED A GRADE OF 100%"
670 PRINT :
    PRINT "KEEP UP THE GOOD WORK!"
680 END
690 :
    '
700 REM  CHOOSE FROM 1 OF 4 'CORRECT' MESSAGES
710 :
    '
720 B = RND(4):
    ON B GOTO 740 ,750 ,760
730 PRINT "NICELY DONE ";N$:
    GOTO 770
740 PRINT "THAT'S RIGHT ";N$:
    GOTO 770
750 PRINT "CORRECT ";N$:
    GOTO 770
760 PRINT "EXCELLENT"
770 C = C + 1:
    RETURN
780 :
    '
790 REM  CHOOSE FROM 1 OF 4 'WRONG' MESSAGES
800 :
    '
810 B = RND(4):
    ON B GOTO 830 ,840 ,850
820 PRINT "SHAME ON YOU ";N$:
    RETURN
830 PRINT "THAT'S WRONG ";N$:
    RETURN
840 PRINT "WHAT HAPPENED ";N$;"?":
    RETURN
850 PRINT "I'M SORRY ";N$:
    RETURN
860 :
    '
870 REM  REM GENERATE NUMERICAL VALUES FOR X,Y,Z
880 :
    '
890 X = RND(30):
    Y = RND(50):
    Z = RND(90):
    RETURN
900 X = .1 * RND(50):
    Y = RND(20)
910 Z = RND(30):
    RETURN
920 :
    '
930 REM  PUT STUDENT NAMES HERE. ADJUST VALUE OF N IN LINE 120
940 :
    '
950 DATA "TOM B","JEFF","MIKE","ANDREW","STEVEN","TOM N","HAN"
960 DATA "MARIA C","CLAUDINE","LISA","DEBBIE","MATTHEW","KEVIN"
970 DATA "MARIA H","JENNIFER","DONNA V","DONNA S","JAIME"
980 DATA "WANDA","PATTY"
990 :
    '
1000 :
    '
1010 :
    '
```

```
1020 :
     '  *    *    *    *    *    *    *    *    *    *    *    *    *    *
     '  *    *         LIST  OF  VARIABLES                          *    *
     '  *    *    *    *    *    *    *    *    *    *    *    *    *    *
1030 :
     '
1040 :
     '  A   =  FLAG FOR INTEGER OR DECIMAL VALUES
1050 :
     '  B   =  FLAG FOR CORRECT OR WRONG MESSAGE
1060 :
     '  C   =  # OF CORRECT ANSWERS GIVEN DURING STUDENT'S TURN
1070 :
     '  C1  =  ACTUAL ANSWER TO PROBLEM
1080 :
     '  D   =  FLAG FOR TYPE OF EQUATION
1090 :
     '  F   =  # OF STUDENTS GIVING ONE OR MORE WRONG ANSWERS
1100 :
     '  H   =  HOLDER FOR STUDENT'S PREVIOUS # CORRECT
1110 :
     ' I,J  =  FOR-NEXT VARIABLES
1120 :
     '  N   =  HOW MANY STUDENTS (ADJUST VALUE IN LINE 120)
1130 :
     '  N$  =  STUDENT'S NAME
1140 :
     '  Q$  =  INPUT VARIABLE
1150 :
     '  R   =  STUDENT'S ANSWER TO PROBLEM
1160 :
     ' S(N)=  SCORE FLAGS (-2 AT START, -1 IF ABSENT, 0-T # CORRECT)
1170 :
     '  T   =  # OF QUESTIONS EACH (ADJUST IN LINE 120)
1180 :
     '  X   =  PROBLEM VALUE (INTEGER 1-30, DECIMAL .1 - 5.0)
1190 :
     '  Y   =  PROBLEM VALUE (INTEGER 1-50)
1200 :
     '  Z   =  PROBLEM VALUE (INTEGER 1-90)
```

Program Listing 2. *Answer-key for decimal sheet (music)*

```
10 :
   '  *    *    *    *    *    *    *    *    *    *    *    *    *    *
20 :
   '  *    *                PROGRAM  TWO                         *    *
30 :
   '  *    *  ANSWER-KEY FOR DECIMAL SHEET (MUSIC)               *    *
40 :
   '  *    *               BY  ANNE  WEISS                       *    *
50 :
   '  *    * ST. PETER'S H.S., NEW BRUNSWICK, NJ 08901 *    *
60 :
   '  *    * GOTO 260 TO SEE SCORES AFTER A BREAK            *    *
65 :
   '  *    * CONNECT AUX CABLE TO SPEAKER FOR SOUND          *    *
68 :
   '  *    * CHANGE VALUE OF AD IN LINE 120 FOR <> 16K *    *
70 :
   '  *    *    *    *    *    *    *    *    *    *    *    *    *    *
80 :
   '
90 :
   ' REM GET MACHINE LANGUAGE PROGRAM INSERTED
```

Program continued

```
100 :
    '
110 DATA 100,75,60,50,60,50:
    FOR I = 1 TO 6:
     READ D:
     NEXT :
    :
    ' ADVANCE DATA POINTER
120 AD = 32738:
    HI = INT(AD / 256):
125 POKE 16527,HI:
     POKE 16526,AD - HI * 256:
     CLEAR:
     :
     ' SET MEM SIZE
130 FOR I = AD TO AD + 28:
     READ DT:
     POKE I,DT:
     NEXT I:
    :
    ' LOAD MACHINE LANGUAGE PART
140 DATA 205,127,10,62,1,14,0,237,91,61,64,69,47,230,3,179,211
150 DATA 255,13,40,4,16,246,24,242,37,32,241,201
160 :
    '
170 REM  INITIALIZE
180 :
    '
190 DIM N$(20),C(15),S(20):
    N = 20:
    P = 15
200 FOR I = 1 TO N:
     S(I) = 1:
     NEXT I
210 FOR I = 1 TO P:
     READ C(I):
     NEXT I:
    REM  READ ANSWERS
220 FOR I = 1 TO N:
     READ N$(I):
     NEXT I:
    REM  READ NAMES
230 :
    '
240 REM  DISPLAY MENU OF STUDENT NAMES AND NUMBERS AND SCORES
250 :
    '
260 IF T = N GOSUB 630 :
    END
270 CLS :
    PRINT "HELLO AGAIN":
    PRINT
280 FOR I = 1 TO N
290  C = S(I) - 1:
     S = INT(100 * C / P + .5)
300  PRINT I;N$(I);S;"%",
310  NEXT I
320 PRINT :
    INPUT "PLEASE ENTER THE NUMBER BEFORE YOUR NAME ";Q
330 IF Q < 1 OR Q > N OR INT(Q) < > Q
     THEN
      320
340 IF S(Q) = 16 PRINT "YOU ALREADY HAVE 100% ";N$(Q):
    GOTO 470
350 :
    '
360 REM  GET STUDENT ANSWERS
370 :
    '
380 CLS :
```

```
    PRINT "OK ";N$(Q);" IT'S TIME TO ENTER YOUR ANSWERS TO THE PROBL
    EMS"
390 PRINT
400 FOR I = S(Q) TO 15:
     REM  START WHERE EACH STUDENT LEFT OFF
410  PRINT I;")   ";:
     INPUT A
420  IF A = C(I)
      THEN
       S(Q) = S(Q) + 1 :
      ELSE
       GOSUB 530
430  C = S(Q) - 1
440  NEXT I:
    PRINT
450 IF S(Q) = 16 PRINT "VERY GOOD":
    S = 100:
    T = T + 1:
    GOSUB 590 :
    REM  MUSIC
460 PRINT N$(Q);" YOU NOW HAVE";C;"CORRECT ANSWERS FOR A SCORE OF";S
    ;"%"
470 PRINT :
    PRINT "PLEASE ASK ANOTHER STUDENT TO COME UP"
480 PRINT :
    PRINT :
    PRINT
490 INPUT "PUSH THE ENTER KEY TO BEGIN ";Q$:
    GOTO 270
500 :
    '
510 REM  WRONG ANSWER ROUTINE
520 :
    '
530 PRINT A; "IS NOT THE ANSWER TO QUESTION #";I
540 SS = USR(256 * 0 + 255):
    REM  BUZZER
550 PRINT "GO BACK TO YOUR SEAT AND TRY AGAIN "
560 S = INT(100 * C / P + .5)
570 I = P:
    RETURN
580 :
    '
590 REM  MELODY FOR 100%
600 :
    '
610 RESTORE :
    FOR I = 1 TO 4:
     READ D:
     SS = USR(256 * 24 + D):
     NEXT I
620 FOR I = 1 TO 32:
     NEXT I
622 FOR I = 1 TO 2:
     READ D:
     SS = USR(256 * 24 + D):
     NEXT I
625 RETURN
627 :
    '
630 REM  EVERYONE GOT 100%
640 :
    '
650 CLS :
    PRINT "VERY GOOD. EVERYONE GOT 100%"
660 PRINT :
    PRINT :
    RETURN
700 :
    '
```

Program continued

```
710 REM  ANSWER KEY. ADJUST VALUE FOR P IN LINE 190
720 :
730 DATA 12.5,23.6,34.7,4.58,5.69
740 DATA 67.1,7.821,8.932,91.043,10.2154
750 DATA 21.3265,32.4376,4.35487,5.46598,65.76109
760 :
770 REM  STUDENT NAMES. ADJUST VALUE FOR N IN LINE 190
780 :
790 DATA "TOM B","JEFF","MICHAEL","ANDREW","STEVEN","TOM N","HAN"
800 DATA "MARIA C","CLAUDINE","DONNA S","DONNA V","JAIME","WANDA","P
    ATTY"
810 DATA "LISA","DEBBIE","MATTHEW","KEVIN","MARIA H","JENNIFER"
820 :
830 :     *  *  *  *  *  *  *  *  *  *  *  *  *  *
840 :     *  *        LIST OF VARIABLES        *  *
850 :     *  *  *  *  *  *  *  *  *  *  *  *  *  *
860 :
870 :     A    =   STUDENT'S ANSWER
880 :     AD   =   DECIMAL ADDRESS FOR 16K MACHINE LANGUAGE PART
890 :     C    =   # CORRECT FOR A GIVEN STUDENT
900 :    C(P)  =   ACTUAL ANSWERS
910 :     D    =   DATA FOR MAKING MUSIC
920 :     DT   =   DATA FOR DRIVING MUSIC PROGRAM
930 :     HI   =   HEX HIGH ADDRESS FOR MACHINE LANGUAGE PART
940 :     I    =   FOR-NEXT VARIABLE
950 :     N    =   HOW MANY STUDENTS (ADJUST VALUE IN LINE 190)
960 :   N$(N)  =   STUDENT NAMES
970 :     P    =   HOW MANY PROBLEMS (ADJUST VALUE IN LINE 190)
980 :     Q    =   INPUT VARIABLE
990 :     Q$   =   INPUT VARIABLE
1000 :   S    =   STUDENT'S SCORE AS A %
1010 :   SS   =   CALL TO SOUND PROGRAM
1020 :  S(N)  =   PROBLEM # TO START WITH FOR EACH STUDENT
1030 :   T    =   TOTAL # OF STUDENTS GETTING 100 %
```

Program Listing 3. *Order of operations and signed numbers*

```
10 :     *  *  *  *  *  *  *  *  *  *  *  *  *  *
20 :
```

```
        *    *           PROGRAM   THREE                  *    *
30 :
   '    *    *  ORDER OF OPERATIONS AND SIGNED NUMBERS    *    *
40 :
   '    *    *           BY    ANNE   WEISS               *    *
50 :
   '    *    * ST. PETER'S H.S., NEW BRUNSWICK, NJ 08901 *    *
60 :
   '    *    * GOTO 540 TO SEE HIGH SCORES AFTER A BREAK *    *
70 :
   '    *    *    *    *    *    *    *    *    *    *    *    *    *    *
80 :
   '
90 REM  INITIALIZATION AND DIRECTIONS
100 :
   '
110 N = 20:
    CLS :
    DIM S(N):
    T = 5
120 PRINT TAB(10),"H E L L O   A G A I N":
    PRINT
130 PRINT "    TODAY WE WILL SEE WHO CAN CORRECTLY ANSWER THE MOST
      QUESTIONS INVOLVING ADDING AND SUBTRACTING SIGNED NUMBERS"
140 PRINT :
    PRINT "YOU WILL EACH HAVE";T;"PROBLEMS TO SOLVE"
150 PRINT :
    PRINT "BE CAREFUL!!!   I'M GOING TO TRY TO TRICK YOU":
    PRINT
160 :
   '
170 REM  TAKE ROLL
180 :
   '
190 PRINT "SOMEONE PLEASE TAKE ATTENDANCE FOR ME":
    PRINT
200 PRINT "ANSWER WITH  Y   OR   N"
210 FOR I = 1 TO N:
     READ N$
220  PRINT N$;:
     INPUT "    ";Q$
230  IF LEFT$(Q$,1) = "N"
      THEN
       S(I) = - 1:
       REM  STUDENT IS ABSENT
240  NEXT I
250 PRINT :
    PRINT "THANK YOU":
    PRINT
260 INPUT "ARE YOU READY TO BEGIN ( Y  OR  N ) ";Q$
270 IF LEFT$(Q$,1) = "N" PRINT "NOW ";:
    GOTO 260
280 IF LEFT$(Q$,1) < > "Y" PRINT "I SAID ";:
    GOTO 260
290 :
   '
300 REM  MAIN LOOP...LINES 320 - 490
310 :
   '
320 FOR L = 1 TO T:
     RESTORE
330  FOR I = 1 TO N:
      READ N$
340   IF S(I) = - 1
       THEN
        480 :
        REM  SKIP ABSENT STUDENTS
350   PRINT :
      PRINT :
      PRINT N$
```

Program continued

```
360   A = RND(5)  :
      :
      ' >
370   X = RND(20):
      B = RND(2):
      IF B = 1
       THEN
        X = - X :
        :
        ' >>GENERATE PROBLEM
380   Y = RND(20)  :
      :
      ' >>> VALUES
390   B = RND(2):
      IF B = 1
       THEN
        Y = (Y + RND(130)) / 10:
        :
        ' >>>AND DISPLAY
400   Z = RND(20):
      B = RND(2):
      IF B = 1
       THEN
        Z = - Z :
        :
        ' >> PROBLEM
410   ON A GOSUB 680 ,700 ,720 ,740 ,760:
      :
      ' >
420   INPUT K
430   IF ABS(K - C)  = > .0001 GOTO 460
440   S(I)  = S(I)  + 1:
      B = RND(5)
450   ON B GOSUB 810 ,820 ,830 ,840 ,850 :
      GOTO 480
460   PRINT N$;", THE CORRECT ANSWER IS ";C
470   B = RND(5):
      ON B GOSUB 860 ,870 ,880 ,890 ,900
480   NEXT I
490  NEXT L:
    C = 0:
    S = T
500 :
    '
510 REM  WINNERS
520 :
    '
530 PRINT :
    PRINT :
    PRINT "ONE MOMENT PLEASE....":
    FOR I = 1 TO 1000:
     NEXT I
540 CLS :
    PRINT "THE WINNER(S)........":
    PRINT
550 RESTORE :
    FOR I = 1 TO 20:
     READ N$
560  IF S(I)  = S PRINT N$:
     C = C + 1
570  NEXT I:
    IF C = 0
     THEN
      S = S - 1:
      IF S > 0 GOTO 550
580 IF S > 0 PRINT :
    PRINT "HAD A SCORE OF "; INT((100 * S / T)  + .5);"%"
590 IF S = 0 PRINT :
    PRINT "    COULD NOT BE FOUND TODAY     NO ONE GOT ANY PROBLEMS C
    ORRECT    BETTER LUCK NEXT TIME!"
600 PRINT :
```

```
    INPUT "WOULD YOU LIKE TO SOLVE MORE PROBLEMS (Y OR N) ";Q$
610 IF LEFT$(Q$,1) = "N"
     THEN
      END
620 FOR I = 1 TO N:
     IF S(I) > - 1
      THEN
       S(I) = 0
630  NEXT I:
    GOTO 320
640 END
650 :
660 REM  PRINT PROBLEMS
670 :
680 C = X + Y + Z
690 PRINT X;"+(";Y;"+ ";Z;") = ";:
    RETURN
700 C = - X - Y + Z
710 PRINT "- ";X;"- ";Y;"+ ";Z;" = ";:
    RETURN
720 C = - X - Y + Z
730 PRINT "-(";X;"+ ";Y;"- ";Z;") = ";:
    RETURN
740 C = - X + Y + Z
750 PRINT "-(";X;"- ";Y;") + ";Z;" = ";:
    RETURN
760 C = - X + Y - Z
770 PRINT "(- ";X;"+";Y;") + (- ";Z;") = ";:
    RETURN
780 :
790 REM  DISPLAY CORRECT OR INCORRECT MESSAGE
800 :
810 PRINT "TERRIFIC!!":
    RETURN
820 PRINT "WAY TO GO!":
    RETURN
830 PRINT "NICELY DONE!":
    RETURN
840 PRINT "GOOD WORK!":
    RETURN
850 PRINT "CORRECT!":
    RETURN
860 PRINT "SHAME ON YOU":
    RETURN
870 PRINT "BETTER STUDY YOUR NEGATIVE NUMBERS":
    RETURN
880 PRINT "YOU KNOW BETTER THAN THAT":
    RETURN
890 PRINT "BETTER LUCK NEXT TIME":
    RETURN
900 PRINT "YOU MUST NOT BE THINKING TODAY !!!":
    RETURN
910 :
920 REM  PUT STUDENT NAMES HERE. ADJUST VALUE FOR N IN LINE 110
930 :
940 DATA "TOM B","JEFF","MICHAEL","ANDREW","STEVEN","TOM N"
950 DATA "HAN","MARIA C","CLAUDINE","DONNA V",DONNA S","JAIME","WAND
    A","PATTY"
960 DATA "LISA","DEBBIE","MATTHEW","KEVIN","MARIA H","JENNIFER"
970 :
980 :
990 :
```

Program continued

```
1000 :
     '    *    *    *    *    *    *    *    *    *    *    *    *    *    *
                       *    *          LIST  OF  VARIABLES             *    *
1010 :
     '    *    *    *    *    *    *    *    *    *    *    *    *    *    *
1020 :
     '
1030 :
     '    A   =  PROBLEM FORMAT FLAG
1040 :
     '    B   =  SIGN, DECIMAL, AND MESSAGE FLAG
1050 :
     '    C   =  ACTUAL ANSWER (ALSO COUNTER FOR WINNER MESSAGE)
1060 :
     '   I,L  =  FOR-NEXT VARIABLES
1070 :
     '    K   =  STUDENT'S ANSWER
1080 :
     '    N   =  HOW MANY STUDENTS (ADJUST VALUE IN LINE 110)
1090 :
     '    N$  =  STUDENT'S NAME
1100 :
     '    Q$  =  INPUT VARIABLE
1110 :
     '    S   =  HIGHEST SCORE ATTAINED
1120 :
     '   S(N)=  # CORRECT FOR EACH STUDENT (-1 IF ABSENT)
1130 :
     '    T   =  NUMBER OF TURNS FOR EACH STUDENT (ADJUST VALUE IN LINE
      110)
1140 :
     '    X   =  PROBLEM VALUE (INTEGER -20 THRU 20, NOT 0)
1150 :
     '    Y   =  PROBLEM VALUE (INTEGER 1-20, DECIMAL .2-15.0)
     '    Z   =  PROBLEM VALUE (INTEGER -20 THRU 20, NOT 0)
```

Program Listing 4. *Solve linear equation* Ax + B = C

```
 10 :
    '    *    *    *    *    *    *    *    *    *    *    *    *    *    *
 20 :
    '     *    *            PROGRAM  FOUR                      *    *
 30 :
    '     *    *   SOLVE LINEAR EQUATIONS AX + B = C           *    *
 40 :
    '     *    *            BY  ANNE  WEISS                    *    *
 50 :
    '     *    * ST. PETER'S H.S., NEW BRUNSWICK, NJ 08901 *    *
 60 :
    '     *    *  GOTO 410 TO DISPLAY SCORES AFTER A BREAK *    *
 70 :
    '    *    *    *    *    *    *    *    *    *    *    *    *    *    *
 80 :
    '
 90 REM  INITIALIZE AND TAKE ROLL
100 :
    '
110 CLS :
    N = 20:
    DIM S(N):
    T = 5
120 PRINT TAB(10),"H E L L O   A G A I N":
    PRINT
130 PRINT "SOMEONE PLEASE TAKE ATTENDANCE FOR ME"
140 PRINT "ANSWER WITH  Y   OR   N"
150 FOR I = 1 TO 20:
```

```
     READ N$
160  PRINT N$;:
     INPUT "    ";Q$
170  IF LEFT$(Q$,1) = "N"
      THEN
       S(I) = - 1
180  NEXT I
190 PRINT :
    PRINT "THANK YOU":
    PRINT
200 INPUT "ARE YOU READY TO BEGIN ( Y  OR  N ) ";Q$
210 IF LEFT$(Q$,1) = "N" PRINT "NOW ";:
    GOTO 200
220 IF LEFT$(Q$,1) < > "Y" PRINT "I SAID ";:
    GOTO 200
230 GOSUB 1190 :
    REM  MENU OF CHOICES
240 :
    '
250 REM  MAIN LOOP...LINES 270 - 360
260 :
    '
270 FOR L = 1 TO T:
     RESTORE
280  FOR I = 1 TO N:
      READ N$
290   IF S(I) = - 1
       THEN
        350 :
        SKIP ABSENT STUDENTS
300   PRINT :
      PRINT :
      PRINT N$
310   PRINT "SOLVE FOR X:      ";:
      ON T5 GOSUB 750 ,830 ,930 ,1050
320   INPUT "X = ";Y:
      D = RND(7)
330   IF ABS(Y - X) < .0001 GOSUB 510 :
       ELSE
        GOSUB 620
340   PRINT
350   NEXT I
360  NEXT L
370 :
    '
380 REM  SCOREBOARD
390 :
    '
400 PRINT "PLEASE WAIT FOR THE SCORE SHEET":
    FOR I = 1 TO 500:
     NEXT I
410 CLS :
    RESTORE :
    FOR I = 1 TO N
420  READ N$:
     IF S(I) > - 1 PRINT N$, INT((100 * S(I)) / T + .5);"%"
430  NEXT I
440 PRINT :
    INPUT "DO YOU WANT TO SOLVE SOME MORE (Y OR N) ";Q$
450 IF LEFT$(Q$,1) < > "Y" CLS :
    PRINT "SO LONG":
    PRINT :
    END
460 FOR I = 1 TO N:
     IF S(I) > - 1
      THEN
       S(I) = 0
470  NEXT I:
    CLS :
    GOTO 230
```

Program continued

```
480 :
490 REM  CORRECT RESPONSE
500 :
510 S(I) = S(I) + 1:
    ON D GOTO 530 ,540 ,550 ,560 ,570 ,580
520 PRINT "NICE WORK ";N$:
    RETURN
530 PRINT "THAT'S RIGHT!":
    RETURN
540 PRINT "WAY TO GO ";N$:
    RETURN
550 PRINT "GOOD WORK!":
    RETURN
560 PRINT "CONGRATULATIONS ";N$:
    RETURN
570 PRINT "TERRIFIC!!!":
    RETURN
580 PRINT "WONDERFUL ";N$:
    RETURN
590 :
600 REM  INCORRECT RESPONSE
610 :
620 ON D GOTO 640 ,650 ,660 ,670 ,680 ,690
630 PRINT "SHAME ON YOU, ";N$:
    GOTO 700
640 PRINT "BETTER STUDY SOME MORE":
    GOTO 700
650 PRINT "THAT'S AWFUL, ";N$:
    GOTO 700
660 PRINT "NO WAY ";N$:
    GOTO 700
670 PRINT "TOO BAD!!!":
    GOTO 700
680 PRINT "NONSENSE!":
    GOTO 700
690 PRINT "I'M SORRY, ";N$
700 PRINT "THE CORRECT ANSWER IS ";X
710 RETURN
720 :
730 REM  GENERATE EASY PROBLEMS
740 :
750 B = RND(20):
    S = RND(2):
    IF S = 2 B = - B
760 C = RND(30):
    S = RND(2):
    IF S = 2 C = - C
770 PRINT "X ";
780 IF B > 0 PRINT "+";
790 PRINT B;"= ";C:
    X = C - B:
    RETURN
800 :
810 REM  GENERATE SLIGHTLY HARDER PROBLEMS
820 :
830 B = RND(30):
    S = RND(2):
    IF S = 2 B = - B
840 C = RND(50):
    S = RND(2):
    IF S = 2 C = - C
```

```
850 S = RND(2):
    IF S = 2 B = (10 * B + RND(9)) * .1
860 S = RND(2):
    IF S = 2 C = (10 * C + RND(9)) * .1
870 PRINT "X ";
880 IF B > 0 PRINT " +";
890 PRINT B;"= ";C:
    X = C - B:
    RETURN
900 :
    '
910 REM  GENERATE HARD PROBLEMS
920 :
    '
930 A = RND(5):
    S = RND(2):
    IF S = 2 A = - A
940 B = RND(50):
    S = RND(2):
    IF S = 2 B = - B
950 C = RND(180):
    S = RND(2):
    IF S = 2 C = - C
960 PRINT "ROUND ALL ANSWERS TO 2 DECIMAL PLACES"
970 IF A = 1 PRINT "X ";:
    GOTO 1000
980 IF A = - 1 PRINT "-X ";:
    GOTO 1000
990 PRINT A;"X ";
1000 IF B > 0 PRINT "+";
1010 PRINT B;"= ";C:
     X = (C - B) / A:
     X = .01 * INT(100 * X + .5):
     RETURN
1020 :
     '
1030 REM  GENERATE REALLY HARD PROBLEMS
1040 :
     '
1050 A = RND(5):
     S = RND(2):
     IF S = 2 A = - A
1060 B = RND(150):
     S = RND(2):
     IF S = 2 B = - B
1070 C = RND(250):
     S = RND(2):
     IF S = 2 C = - C
1080 S = RND(2):
     IF S = 2 B = .1 * B
1090 S = RND(2):
     IF S = 2 C = .1 * C
1100 PRINT "ROUND ALL ANSWERS TO 2 DECIMAL PLACES"
1110 IF A = 1 PRINT "X ";:
     GOTO 1140
1120 IF A = - 1 PRINT "-X ";:
     GOTO 1140
1130 PRINT A;"X ";
1140 IF B > 0 PRINT "+";
1150 PRINT B;"= ";C:
     X = (C - B) / A:
     X = .01 * INT(100 * X + .5):
     RETURN
1160 :
     '
1170 REM  MENU OF CHOICES
1180 :
     '
1190 CLS :
```

Program continued

```
     PRINT "DO YOU WISH TO SOLVE":
     PRINT
1200 PRINT "1.    EASY PROBLEMS SUCH AS  X + 12 = 5"
1210 PRINT "2.    SLIGHTLY HARDER ONES SUCH AS   X + 22.9 = 47.1"
1220 PRINT "3.    HARDER ONES SUCH AS   5X - 3 = -8"
1230 PRINT "4.    REALLY HARD ONES SUCH AS   -3X + 17.4 = 94.1"
1240 PRINT :
     INPUT "ENTER 1, 2, 3, OR 4  PLEASE ";T5
1250 IF T5 < > 1 AND T5 < > 2 AND T5 < > 3 AND T5 < > 4 GOTO 1240
1260 RETURN
1270 :
     '
1280 REM  PUT STUDENT NAMES HERE. ADJUST VALUE FOR N IN LINE 110
1290 :
     '
1300 DATA "TOM B","JEFF","MICHAEL","ANDREW","STEVEN","TOM N"
1310 DATA "HAN","MARIA C","CLAUDINE","DONNA S","JAIME","WANDA","PATTY
     "
1320 DATA "LISA","DEBBIE","MATTHEW","KEVIN","MARIA H","JENNIFER","DON
     NA V"
1330 :
     '
1340 :
     '  *   *   *   *   *   *   *   *   *   *   *   *   *   *
1350 :
     '  *   *        LIST  OF  VARIABLES                *   *
1360 :
     '  *   *   *   *   *   *   *   *   *   *   *   *   *   *
1370 :
     '
1380 :
     '   A   =  PROBLEM VALUE (INTEGER -5 THROUGH 5)
1390 :
     '   B   =  PROBLEM VALUE (INTEGER 1-50,DECIMAL .1-30.9, + OR -)
1400 :
     '   X   =  ACTUAL ANSWER
1410 :
     '   Y   =  STUDENT'S ANSWER
1420 :
     ' I,L   =  FOR-NEXT VARIABLES
1430 :
     '   N   =  HOW MANY STUDENTS (ADJUST VALUE IN LINE 110)
1440 :
     '   N$  =  STUDENT'S NAME
1450 :
     '   Q$  =  INPUT VARIABLE
1460 :
     '   S   =  SIGN FLAG
1470 :
     ' S(N)=  # CORRECT (-1 IF ABSENT)
1480 :
     '   t   =  NUMBER OF TURNS (ADJUST VALUE IN LINE 110
1490 :
     '   T5  =  TYPE OF PROBLEM FLAG
```

Program Listing 5. *Decimal linear equations with brackets*

```
10 :
   '  *   *   *   *   *   *   *   *   *   *   *   *   *   *
20 :
   '  *   *              PROGRAM  FIVE                 *   *
30 :
   '  *   *  DECIMAL LINEAR EQUATIONS WITH BRACKETS    *   *
40 :
   '  *   *              BY  ANNE  WEISS               *   *
50 :
   '  *   * ST PETER'S H.S., NEW BRUNSWICK, NJ 08901   *   *
60 :
   '
```

```
      *   *    GOTO 570 TO SEE SCORES AFTER A BREAK    *   *
 70 :
    '   *   *   *   *   *   *   *   *   *   *   *   *   *   *
 80 :
    '
 90 REM  INITIALIZE
100 :
    '
110 CLS :
    CLEAR 3000
120 I$ = " PUSH ENTER FOR YOUR EQUATION"
130 PRINT @ 340,"H E L L O":
    FOR I = 1 TO 500:
     NEXT
140 PRINT @ 640, "WELCOME TO THE ALGEBRA ROUND ROBIN"
150 FOR I = 1 TO 1000:
     NEXT :
    CLS
160 PRINT "UP TO 6 STUDENTS OR TEAMS MAY PARTICIPATE":
    PRINT
170 INPUT "HOW MANY STUDENTS (OR TEAMS) ARE ENTERING THE CONTEST ";N
180 IF N < 1 OR N > 6 OR INT(N) < > N
     THEN
      CLS :
      GOTO 160
190 DIM S(N),N$(N),E$(24),A(24):
    PRINT
200 FOR I = 1 TO 24:
     READ E$(I),A(I):
     NEXT
210 FOR I = 1 TO N:
     S(I) = 0
220  PRINT "ENTER THE NAME OF STUDENT (OR TEAM) #";I;:
     INPUT N$(I)
230  PRINT :
     NEXT I:
    CLS
240 PRINT "O.K.   HERE WE GO ....."
250 PRINT "I WILL GIVE YOU AN EQUATION TO SOLVE"
260 PRINT "YOU WILL HAVE ABOUT 2 MINUTES TO ENTER YOUR ANSWER"
270 PRINT :
    INPUT "PUSH ENTER TO START ";Q$:
    CLS
280 :
    '
290 REM  MAIN LOOP...LINES 310 - 550
300 :
    '
310 FOR J = 1 TO 4
320  PRINT :
     PRINT :
     PRINT TAB(25);"R O U N D   # ";J
330  PRINT
340  FOR I = 1 TO N:
      X = RND(24)
350   IF E$(X) = ""
       THEN
        X = X + 1:
        IF X = 25
         THEN
          X = 1:
          :
          '  KEEP GOING UNTIL AN
360   IF E$(X) = ""
       THEN
        350 :
        :
        '  UNANSWERED ONE IS FOUND
370   PRINT E$(X):
      PRINT "WHAT IS YOUR ANSWER ";N$(I)
```

Program continued

```
380    PRINT "X = ";:
       C = 0:
       Y = RND(5) + 5:
       H$ = ""
390    A$ = INKEY$:
       IF A$ = ""
        THEN
         C = C + 1:
         IF C = 5000
          THEN
           650 :
          ELSE
           390
400    IF ASC(A$) < 45 OR ASC(A$) > 57
        THEN
         390 :
        ELSE
         PRINT A$;:
         H$ = A$
410    B$ = INKEY$:
       IF B$ = ""
        THEN
         C = C + 1:
         IF C = 5000
          THEN
           650 :
          ELSE
           410
420    IF ASC(B$) = 13
        THEN
         450
430    IF ASC(B$) = 8
        THEN
         H$ = LEFT$(H$, LEN(H$) - 1):
         PRINT B$;:
         IF H$ < > ""
          THEN
           410 :
          ELSE
           390
440    IF ASC(B$) < 45 OR ASC(B$) > 57
        THEN
         410 :
        ELSE
         PRINT B$;:
         H$ = H$ + B$:
         GOTO 410
450    PRINT :
       A = VAL(H$)
460    IF A = A(X)
        THEN
         E$(X) = "":
         Y = Y - 5:
         :
         '  FLAG QUESTION AS ANSWERED CORRECTLY
470    ON Y GOSUB 690 ,700 ,710 ,720 ,730 ,770 ,780 ,790 ,800 ,810
480    X = I + 1:
       IF X = N + 1
        THEN
         X = 1
490    IF X = 1 AND J = 4
        THEN
         FOR K = 1 TO 1000:
          NEXT :
         GOTO 530
500    PRINT @832,N$(X);I$:
       :
       '  SIGNAL THE NEXT STUDENT
510    FOR K = 1 TO 100:
        NEXT :
```

```
     PRINT @896,"ENTER";:
     Q$ = INKEY$
520  IF Q$ = ""
      THEN
       FOR K = 1 TO 100:
        NEXT :
       PRINT @896,"     ";:
       GOTO 510
530  CLS
540  NEXT I
550 NEXT J
560 :
    '
570 REM  SCORES
580 :
    '
590 PRINT "THE SCORES ARE":
    PRINT
600 FOR I = 1 TO N:
     PRINT N$(I),25 * S(I):
     NEXT I:
    PRINT
610 PRINT :
    END
620 :
    '
630 REM  TIME RAN OUT
640 :
    '
650 PRINT :
    PRINT "SORRY ";N$(I);" YOU TOOK TOO LONG":
    GOTO 480
660 :
    '
670 REM  CORRECT RESPONSE
680 :
    '
690 PRINT "WAY TO GO ";N$(I):
    S(I) = S(I) + 1:
    RETURN
700 PRINT "RIGHT ON!":
    S(I) = S(I) + 1:
    RETURN
710 PRINT "THAT'S GOOD, ";N$(I):
    S(I) = S(I) + 1:
    RETURN
720 PRINT "TERRIFIC!!":
    S(I) = S(I) + 1:
    RETURN
730 PRINT "GOOD WORK, ";N$(I):
    S(I) = S(I) + 1:
    RETURN
740 :
    '
750 REM  INCORRECT RESPONSE
760 :
    '
770 PRINT "SORRY ";N$(I):
    RETURN
780 PRINT "BETTER LUCK NEXT TIME":
    RETURN
790 PRINT "SHAME ON YOU, ";N$(I):
    RETURN
800 PRINT "W R O N G ! ":
    RETURN
810 PRINT "TOO BAD ";N$(I):
    RETURN
820 :
    '
830 REM  24 PROBLEMS AND ANSWERS
```

Program continued

```
 840 :
     '
 850 DATA ".07X + .04(9000 - X) = 450",3000
 860 DATA ".06X - .04(3500 - X) = 160",3000
 870 DATA ".06(X - 5) = .04(X + 8)",31
 880 DATA ".04X + .03(2000 - X) = 75",1500
 890 DATA ".13X - 1.4 = .08X + 7.6",180
 900 DATA ".05X - .25 = .02X + .44",23
 910 DATA ".8X + 2.6 = .2X + 9.8",12
 920 DATA ".06X + 40 - .03X = 70",1000
 930 DATA ".02(X + 5) = 8",395
 940 DATA ".05(X - 8) = .07X",-20
 950 DATA ".4(X - 9) = .3(X + 4)",48
 960 DATA ".02X + .04(1500 - X) = 48",600
 970 DATA ".05X + 10 = .06(X + 50)",700
 980 DATA ".08X = .03(X + 200) - 4",40
 990 DATA "1.7X = 30 + .2X",20
1000 DATA "1.5X - 1.69 = .2X",1.3
1010 DATA ".08X = 1.5 + .07X",150
1020 DATA ".5X - .3X = 8",40
1030 DATA "2X + .5X = 50",20
1040 DATA ".08X - .9 = .02X",15
1050 DATA ".7X - .4 = 1",2
1060 DATA ".03X - 1.2 = 8.7",330
1070 DATA ".4X + .08 = 4.24",10.4
1080 DATA ".05X + .04(500 - X) = 22",200
1090 :
     '
1100 :
     '

          *   *   *   *   *   *   *   *   *   *   *   *   *
1110 :
     '    *   *           LIST  OF  VARIABLES         *   *
1120 :
     '    *   *   *   *   *   *   *   *   *   *   *   *   *
1130 :
     '
1140 :
     '  A() =  24 ACTUAL ANSWERS
1150 :
     '   A  =  STUDENT'S FINAL ANSWER
1160 :
     '   A$ =  FIRST INKEY$ OF ANSWER
1170 :
     '   B$ =  REST OF INKEY$ OF ANSWER
1180 :
     '   C  =  TIMER COUNTER (5000 GIVES 2 MINUTES - LINES 390 & 410)
1190 :
     '  E$()=  24 EQUATIONS
1200 :
     '   H$ =  HOLDS ANSWER UNTIL DONE ENTERING
1210 :
     '   I$ =  CONTINUATION MESSAGE
1220 :
     '  I,J =  FOR-NEXT VARIABLES
1230 :
     '   N  =  HOW MANY PARTICIPATING
1240 :
     '  N$()=  NAMES OF STUDENTS OR TEAMS
1250 :
     '   Q$ =  INPUT VARIABLE
1260 :
     '  S() =  # CORRECT FOR EACH STUDENT
1270 :
     '   X  =  RANDOM FLAG FOR CHOOSING EQUATION
1280 :
     '   Y  =  RANDOM FLAG FOR CHOOSING MESSAGE DISPLAYED
```

Program Listing 6. *Randomly assigned problems*

```
10 :
   '   *   *   *   *   *   *   *   *   *   *   *   *   *   *
20 :
   '   *   *               PROGRAM SIX                 *   *
30 :
   '   *   *   STUDENTS DO RANDOMLY ASSIGNED PROBLEMS  *   *
40 :
   '   *   *   ON ONE OF FIVE ASSIGNED WORKSHEETS      *   *
50 :
   '   *   *             BY  ANNE  WEISS               *   *
60 :
   '   *   * ST PETER'S H.S., NEW BRUNSWICK, NJ 08901  *   *
70 :
   '   *   *     GOTO 600 FOR SCORES AFTER A BREAK     *   *
80 :
   '   *   *   *   *   *   *   *   *   *   *   *   *   *   *
90 :
   '
100 REM  INITIALIZE AND GIVE DIRECTIONS
110 :
    '
120 CLS :
    PRINT "H E L L O   S T U D E N T S":
    PRINT
130 PRINT "TODAY YOU WILL DO PROBLEMS FROM A WORKSHEET"
140 PRINT "YOUR TEACHER WILL TELL YOU IF YOU ARE TO WORK ALONE OR IN
     TEAMS."
150 PRINT "UP TO 6 GROUPS MAY PARTICIPATE":
    PRINT
160 INPUT "HOW MANY STUDENTS (OR TEAMS) ARE THERE ";N
170 IF N < 7 AND N > 0 AND INT(N) = N
     THEN
      190
180 PRINT "PLEASE COUNT AGAIN...MAXIMUM IS 6":
    PRINT :
    GOTO 160
190 DIM N$(N),H(N,5),A(30),S(N),C(30),F(30):
    PRINT :
    PRINT
200 PRINT "ASK YOUR TEACHER WHICH WORKSHEET YOU ARE USING TODAY"
210 INPUT "ENTER THE NUMBER OF THAT WORKSHEET ";W
220 IF W < 6 AND W > 0 AND INT(W) = W
     THEN
      240
230 PRINT "DON'T BE A SMART-ALECK!!!":
    PRINT :
    GOTO 210
240 CLS
250 FOR I = 1 TO N
260  PRINT "ENTER THE NAME OF THE STUDENT (OR TEAM) #";I;:
     INPUT N$(I)
270  NEXT I
280 :
    '
290 REM  ASSIGN PROBLEMS
300 :
    '
310 FOR I = 1 TO W:
     FOR J = 1 TO 30:
      READ A(J):
      NEXT J:
     NEXT I:
    CLS
320 FOR I = 1 TO N
330  PRINT N$(I);" DO PROBLEM #",
340  FOR J = 1 TO 5:
      X = RND(30)
```

Program continued

```
350   IF F(X) = 1
       THEN
        X = X + 1:
        IF X = 31
         THEN
          X = 1
360   IF F(X) = 1
       THEN
        350
370   PRINT X;:
      H(I,J) = X:
      F(X) = 1:
      NEXT J
380  PRINT :
     PRINT :
     NEXT I
390 :
    '
400 REM  GET ANSWERS
410 :
    '
420 C = 0:
    PRINT :
    INPUT "PUSH ENTER WHEN ANSWERS ARE READY TO BE ENTERED ";Q$
430 FOR I = 1 TO N:
     IF S(I) = 5
      THEN
       C = C + 1:
       GOTO 560
440  CLS :
     PRINT "OK ";N$(I);", WHAT  ARE YOUR ANSWERS?"
450  PRINT "JUST PUSH ENTER FOR THOSE YOU DON'T KNOW YET":
     PRINT
460  FOR J = 1 TO 5:
      X = H(I,J):
      Q$ = ""
470   IF C(X) = 1
       THEN
        520
480   Q = - 99999.99:
      PRINT "#";X;:
      INPUT "   ";Q
490   IF Q = - 99999.99 PRINT "YOU CAN TRY THAT ONE LATER ";N$(I):
      GOTO 520
500   R = RND(5):
      IF Q = A(X)
       THEN
        R = R + 5:
        S(I) = S(I) + 1:
        C(X) = 1
510   ON R GOSUB 690 ,700 ,710 ,720 ,730 ,740 ,750 ,760 ,770 ,780
520   NEXT J:
     PRINT
530  ON 1 + S(I) GOSUB 820 ,830 ,840 ,850 ,860 ,870
540  PRINT :
     IF I < N INPUT "PUSH ENTER TO CONTINUE ";Q$
550  IF N = I INPUT "PUSH ENTER TO SEE ALL SCORES ";Q$
560  NEXT I:
    PRINT
570 :
    '
580 REM  DISPLAY SCORES
590 :
    '
600 FOR I = 1 TO N:
     PRINT N$(I),20 * S(I):
     NEXT I
610 :
    '
620 REM  SEE IF ANY PROBLEMS NOT SOLVED YET
```

```
630 :
640 PRINT :
    PRINT :
    IF C = N END
650 PRINT :
    GOTO 420
660 :
670 REM  DISPLAY 'CORRECT'  OR  'INCORRECT'  MESSAGE
680 :
690 PRINT "TOO BAD - THAT'S WRONG!":
    RETURN
700 PRINT "SORRY ";N$(I);".  bETTER CHECK YOUR WORK":
    RETURN
710 PRINT "SHAME ON YOU - YOU CAN DO BETTER THAN THAT":
    RETURN
720 PRINT "H O R R O R S, ";N$(I);"!  I THOUGHT YOU KNEW ALGEBRA":
    RETURN
730 PRINT "NOW REALLY!!!  TRY AGAIN":
    RETURN
740 PRINT "CORRECT ";N$(I):
    RETURN
750 PRINT "T E R R I F I C":
    RETURN
760 PRINT "WAY TO GO, ";N$(I):
    RETURN
770 PRINT "KEEP UP THE GOOD WORK":
    RETURN
780 PRINT "WONDERFUL !!!":
    RETURN
790 :
800 REM  SCOREBOARD MESSAGES
810 :
820 PRINT "SHAME ON YOU ";N$(I);".  yOU DIDN'T GET ANY CORRECT":
    GOTO 880
830 PRINT "YOU CAN DO BETTER THAN ONLY 1 CORRECT, ";N$(I):
    GOTO 880
840 PRINT "40% IS NOT TOO GOOD ";N$(I):
    GOTO 880
850 PRINT N$(I);", SO FAR YOU HAVE A SCORE OF 60%":
    GOTO 880
860 PRINT "VERY GOOD, ";N$(I);"!  YOU NOW HAVE A SCORE OF 80%":
    GOTO 880
870 CLS :
    PRINT "CONGRATULATIONS, "N$(I);"!  YOU EARNED 100%":
    RETURN
880 PRINT "GO BACK AND DO THE PROBLEMS YOU MISSED"
890 PRINT "YOU WILL HAVE ANOTHER CHANCE TO ENTER YOUR ANSWERS"
900 RETURN
910 :
920 REM  ANSWERS FROM WORKSHEET ONE
930 :
940 DATA 7.1,4.7,2500,40,50,3,500,2,80,.02,3.1,2,2.7,5,2.26
950 DATA .3,6.65,2,3.27,8,2.52,7.4,.5,60,13.4,50,10.5,10,5.4,.05
960 :
970 REM  ANSWERS FROM WORKSHEET TWO
980 :
990 DATA 3,.8,1.5,4,16,.3,100,10,200,5
1000 DATA 10,105,500,6,.2,6,10,10,.4,12,2,22,-12,200
1010 DATA 20,200,300,300,10,600
1020 :
1030 REM  30 ANSWERS FROM WORKSHEET THREE
```

Program continued

```
1040 :
     '
1050 :
     '
1060 REM  30 ANSWERS FROM WORKSHEET FOUR
1070 :
     '
1080 REM  30 ANSWERS FROM WORKSHEET FIVE
1090 :
     '
1100 :
     '
1110 :
     '    *   *   *   *   *   *   *   *   *   *   *   *   *
1120 :
     '    *   *           LIST  OF  VARIABLES         *   *
1130 :
     '    *   *   *   *   *   *   *   *   *   *   *   *   *
1140 :
     '
1150 :
     '  A(30)  =   ACTUAL ANSWERS
1160 :
     '    C    =   HOW MANY GOT 100%
1170 :
     '  C(30)  =   FLAG FOR WHICH PROBLEMS HAVE BEEN CORRECTLY ANSWERE
     D
1180 :
     '  F(30)  =   FLAG FOR WHICH PROBLEMS ALREADY ASSIGNED
1190 :
     '  H(N,5)=   PROBLEMS ASSIGNED EACH STUDENT
1200 :
     '   I,J   =   FOR-NEXT VARIABLES
1210 :
     '    N    =   NUMBER PARTICIPATING
1220 :
     '  N$(N)  =   PARTICIPANTS' NAMES
1230 :
     '    Q    =   STUDENT'S ANSWER
1240 :
     '    Q$   =   INPUT VARIABLE
1250 :
     '    R    =   FLAG FOR WHICH MESSAGE DISPLAYED
1260 :
     '   S(N)  =   SCORES
1270 :
     '    X    =   RANDOM SELECTOR FOR PROBLEMS
1280 :
     '    W    =   WORKSHEET NUMBER
```

GAMES

Supermaze

Micro Basketball

Supermaze

by Howard F. Batie

Supermaze puts you inside a maze with a corridor ahead of you in complete perspective. Halls lead off to the right and left (see Figure 1). It's up to you to guess which way to go. You can see a maximum distance of four units ahead. If there's a wall three squares ahead, however, you can't see beyond it.

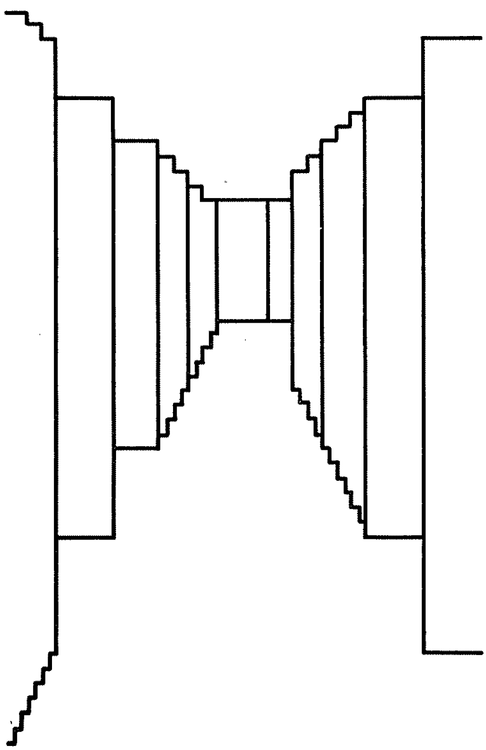


Figure 1

Three Options

Each move offers three options: forward one space, left, or right. After each move, you get a new picture of what lies ahead. A counter keeps track of the number of forward moves, but is not incremented if you turn. The minimum number of forward moves needed to exit successfully and your score are printed on the screen when you leave the maze. If you get turned

around and leave the maze at the entrance, you lose. And if you're unfortunate enough to walk into an electric wall, you fry.

The Program Listing contains six mazes of increasing difficulty and is written for a Level II TRS-80 with 16K. The program's first array is called A and has the dimensions of 105 × 1. It uses the zero element. The first 100 elements (0–99) contain either a zero value or a five-digit decimal number which defines the shape of the element's maze location. Visualize the first 100 elements of the A array as a 10 × 10 matrix (which is the maximum size of the maze) as shown in Figure 2.

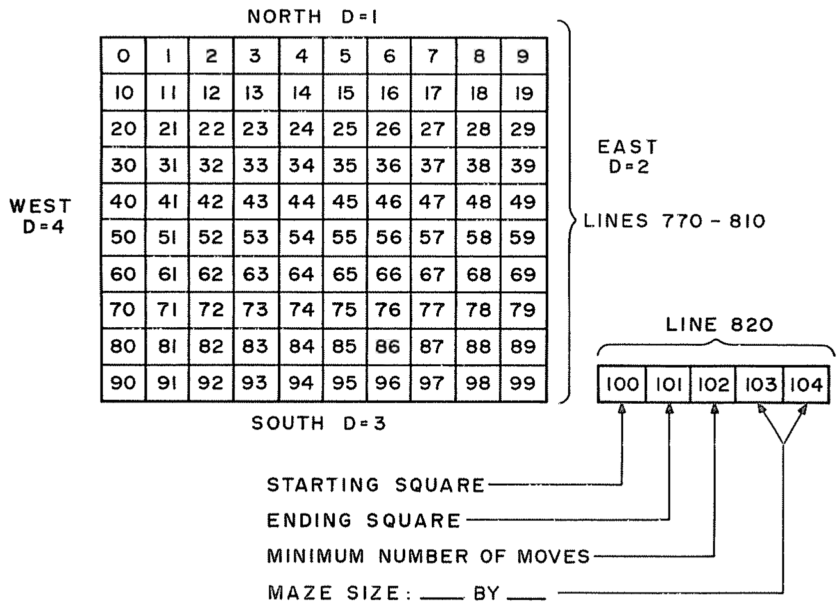


Figure 2

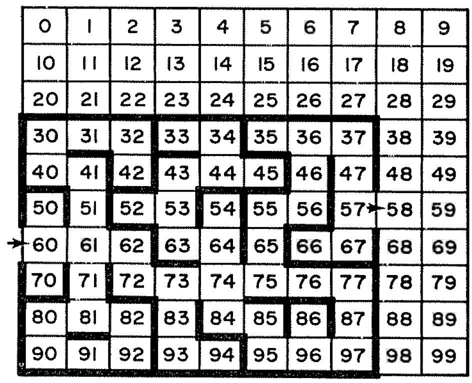
Constructing the Maze

The Program Listing statements 930–980 construct the 8 × 7 maze of Figure 3, with the entrance at location 60 and the exit at location 57. Other mazes shown in Figures 4 through 8 are constructed by statements 990–1270. Note that all mazes must be entered from the left side and exited on the right side, because the initial direction (D) is equal to 2 in line 200.

The final five elements of the A array (100–104) specify the starting and ending locations, minimum number of moves to the exit, and the size of the maze. These can differ for each maze. If any numbered matrix location in the grid is outside the maze, the contents of the corresponding element will be set to zero; otherwise, the five-digit decimal defines the shape of the maze

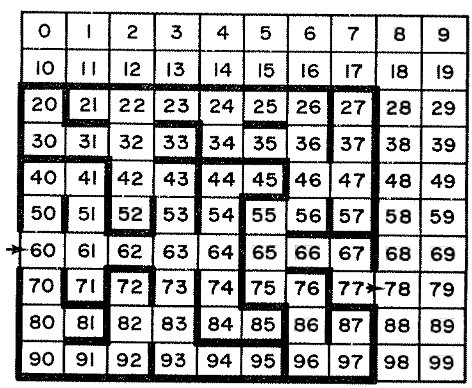
location. To prevent blanking of the leading zeros in the last four digits, the first of the five digits is always one.

In each of the last four digits, a one represents a wall, and a zero represents a hall (no wall). The second, third, fourth, and fifth digits correspond to the north, east, south, and west sides. For example, the shape of block 60 in Figure 3 is designated by 10000, and block 65 is designated by 10101.



8 x 7
14 MOVES

Figure 3

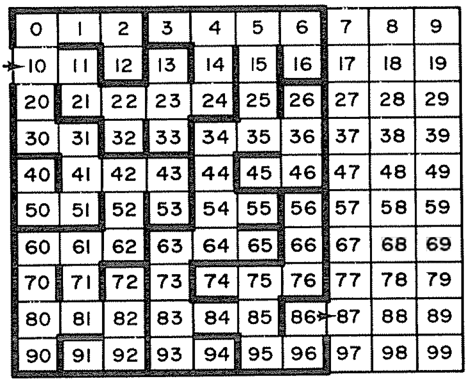


8 x 8
20 MOVES

Figure 4

Changing the Shape

To change the shape of the maze, simply code the data statements to correspond with the particular maze you construct. Lines 990–1040 and 1050–1100 in Program Listing 1 correspond to the mazes of Figures 4 and 5.

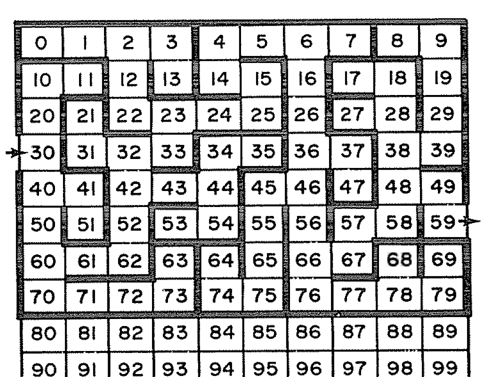


7 x 10
23 MOVES

Figure 5

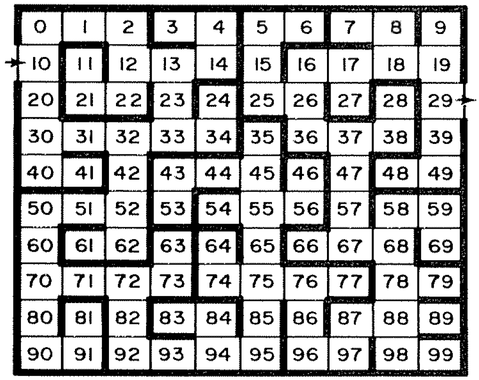
After you have created a number of mazes, video prompts let you choose one of the six mazes to replay and run.

Of course, nearly every program written can be refined, and this one is no exception. Two improvements that come to mind are a built-in random maze generator and the use of machine-language graphics which would provide more speed.



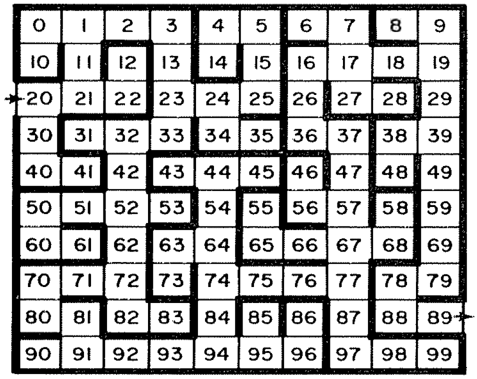
10 x 8
29 MOVES

Figure 6



10 x 10
38 MOVES

Figure 7



10 x 10
45 MOVES

Figure 8

Program Listing

(Please note: Because of space considerations, this program has not been formatted.)

```
5 REM --- SUPERMAZE FOR THE TRS-80 LEVEL II 16K
10 REM -- VERSION 6.2
15 REM -- COPYRIGHT BY HOWARD F. BATIE 1980
25 REM -- DISPLAY TITLE
30 CLS:PRINT@330,"*  *  *   S  U  P  E  R  M  A  Z  E   *  *  *"
40 PRINT@704,"COPYRIGHT 1980"
50 PRINT"BY HOWARD F. BATIE"
60 PRINT"HERNDON, VA"
70 FORI=1TO1000:NEXTI
75 REM -- SETUP
80 CLEAR:DIMA(104),D(23)
90 FORI=0TO23:READD(I):NEXTI
110 CLS:PRINT"   WHICH MAZE DO YOU WANT"
120 LL=0:FORI=0TO5:PRINT@128*I+197,"MAZE NR";D(4*I+LL);
130 PRINT"IS";D(4*I+LL+1);"BY";D(4*I+LL+2);
140 PRINTTAB(26),"MINIMUM NUMBER OF MOVES IS";D(4*I+LL+3)
142 LL=0:NEXTI
150 PRINT@25,"";:INPUTMN
160 IFMN>6ORMN<1PRINT@25,"     ";:GOTO150
170 CLS:IFMN=1THEN190
180 FORI=0TO(105*(MN-1))-1:READAA:NEXTI
190 FORI=0TO104:READA(I):NEXTI
200 E=0:X=A(100):D=2
205 REM -- INSTRUCTIONS
210 CLS:PRINT@128,"YOU ARE IN A";A(103);"BY";A(104);
212 PRINT"MAZE WITH ELECTRIFIED WALLS.  FIND YOUR"
220 PRINT"WAY OUT IN THE LEAST NUMBER OF MOVES.":PRINT
230 PRINT"THE MINIMUM NUMBER OF MOVES IS";A(102);
232 PRINT"FOR THIS MAZE.":PRINT
240 PRINT"MOVE FORWARD BY TYPING 'F', TURN RIGHT BY TYPING 'R'"
250 PRINT"OR TURN LEFT BY TYPING 'L'.":PRINT
260 PRINT"TURNS TO THE RIGHT OR LEFT DO NOT COUNT AS MOVES."
270 PRINT:PRINT"PRESS 'ENTER' WHEN READY TO START."
280 M$=INKEY$:IFM$=""THEN280
285 REM -- START
290 CLS:GOSUB830
295 REM -- IS THERE A WALL TO THE RIGHT?
300 PRINT@435,"MOVES:";:PRINT@500,Q;
301 ONE+1GOTO302,307,310,313,317
302 GL=15374:GR=15399
303 POKEGR,160:POKEGL,144
304 GR=GR+64:GL=GL+64:IFGR>16240THEN306
305 POKEGR,170:POKEGL,149:GOTO304
306 POKE16270,133:POKE16295,138:GOTO321
307 GL=15442:GR=15459
308 GL=GL+64:GR=GR+64:IFGR>16100THEN321
309 POKEGL,149:POKEGR,170:GOTO308
310 GL=15509:GR=15520
311 GL=GL+64:GR=GR+64:IFGR>15970THEN321
312 POKEGL,149:POKEGR,170:GOTO311
313 GL=15575:GR=15582:POKEGL,144:POKEGR,160
314 GL=GL+64:GR=GR+64:IFGR>15840THEN316
315 POKEGL,149:POKEGR,170:GOTO314
316 POKEGL,133:POKEGR,138:GOTO321
317 GL=15641:GR=15644
318 POKEGL,148:POKEGR,168:POKE15833,129:POKE15836,130
319 GL=GL+64:GR=GR+64:IFGR>15773THEN321
320 POKEGL,149:POKEGR,170:GOTO319
321 IFVAL(MID$(B$,4,1))=0THEN339
322 REM --- DRAW WALL TO THE RIGHT
323 ONE+1GOTO324,326,330,333,335
324 POKE15400,184:POKE15401,142:POKE15402,131:POKE16296,180
325 POKE16360,130:POKE16361,173:POKE16362,144:GOTO360
326 POKE15524,131:POKE15460,160:POKE15461,184
327 POKE15462,142:POKE15463,171:POKE16100,144
```

```
328 POKE16164,139:POKE16165,180:POKE16229,130
329 POKE16230,173:POKE16231,186:POKE16295,139:GOTO360
330 POKE15585,131:POKE15521,160:POKE15522,184
331 POKE15523,174:POKE15969,144:POKE16033,139
332 POKE16034,180:POKE16098,130:POKE16099,175:GOTO360
333 POKE15583,184:POKE15584,174:POKE15903,180
334 POKE15967,130:POKE15968,175:GOTO360
335 POKE15645,142:POKE15646,171
336 POKE15836,130:POKE15837,173:POKE15838,186
337 POKE15902,139:GOTO360
338 REM --- DRAW HALL TO THE RIGHT
339 ONE+1GOTO340,343,346,348,350
340 GT=15399:GB=16295
341 GT=GT+1:GB=GB+1:IFGB=16301THEN360
342 POKEGT,176:POKEGB,140:GOTO341
343 POKE15524,131:POKE15525,131:POKE15526,131
344 POKE15527,171:POKE16100,176:POKE16101,176
345 POKE16102,176:POKE16103,186:GOTO360
346 POKE15585,131:POKE15586,131:POKE15587,171
347 POKE15969,176:POKE15970,176:POKE15971,186:GOTO360
348 POKE15583,176:POKE15584,186:POKE15903,140
349 POKE15904,174:GOTO360
350 POKE15645,140:POKE15646,174:POKE15837,131
351 POKE15838,171
355 REM -- IS THERE A WALL TO THE LEFT?
360 IFVAL(MID$(B$,6,1))=0THEN377
361 REM --- DRAW WALL TO THE LEFT
362 ONE+1GOTO363,366,370,373,375
363 POKE15373,180:POKE15372,141:POKE15371,131
364 POKE16269,184:POKE16333,129:POKE16332,158
365 POKE16331,160:GOTO420
366 POKE15438,151:POKE15439,141:POKE15440,180
367 POKE15441,144:POKE15505,131:POKE16270,135
368 POKE16206,181:POKE16207,158:POKE16208,129
369 POKE16144,184:POKE16145,135:POKE16081,160:GOTO420
370 POKE15506,157:POKE15507,180:POKE15508,144
371 POKE15572,131:POKE16082,159:POKE16083,129
372 POKE16019,184:POKE16020,135:POKE15956,160:GOTO420
373 POKE15573,157:POKE15574,180:POKE15957,159
374 POKE15958,129:POKE15894,184:GOTO420
375 POKE15639,151:POKE15640,141:POKE15895,135
376 POKE15831,181:POKE15832,158:GOTO420
377 REM --- DDRAW HALL TO THE LEFT
378 ONE+1GOTO379,391,394,396,398
379 GT=15369:GB=16265
380 GT=GT+1:GB=GB+1:IFGB=16270THEN420
390 POKEGT,176:POKEGB,140:GOTO380
391 POKE15502,151:POKE15503,131:POKE15504,131
392 POKE15505,131:POKE16078,181:POKE16079,176
393 POKE16080,176:POKE16081,176:GOTO420
394 POKE15570,151:POKE15571,131:POKE15572,131
395 POKE15954,181:POKE15955,176:POKE15956,176:GOTO420
396 POKE15573,181:POKE15574,176:POKE15893,157
397 POKE15894,140:GOTO420
398 POKE15639,157:POKE15640,140:POKE15831,151
399 POKE15832,131
415 REM --- IS THERE A WALL AHEAD?
430 IF((X+(E*Y)=A(100))*(D=4))+((X+(E*Y)=A(101))*(D=2))THEN500
440 IFVAL(MID$(B$,3,1))=1THEN460
450 GOTO481
455 REM --- DRAW WALL AHEAD
460 ONE+1GOTO461,465,469,473,477
461 POKE16270,141:POKE16295,142:POKE15374,176
462 POKE15399,176:GT=15374:GB=16270
463 GT=GT+1:GB=GB+1:IFGT=15399THEN500
464 POKEGT,176:POKEGB,140:GOTO463
465 POKE15506,151:POKE15523,171:POKE16082,181
466 POKE16099,186:GT=15506:GB=16082
467 GT=GT+1:GB=GB+1:IFGT=15523THEN500
```

Program continued

```
468 POKEGT,131:POKEGB,176:GOTO467
469 POKE15573,151:POKE15584,171:POKE15957,181
470 POKE15968,186:GT=15573:GB=15957
471 GT=GT+1:GB=GB+1:IFGT=15584THEN500
472 POKEGT,131:POKEGB,176:GOTO471
473 POKE15575,176:POKE15582,176:POKE15895,141
474 POKE15902,142:GT=15575:GB=15895
475 GT=GT+1:GB=GB+1:IFGT=15582THEN500
476 POKEGT,176:POKEGB,140:GOTO475
477 GT=15641:GB=15833
478 POKEGT,156:POKEGB,131
479 GT=GT+1:GB=GB+1:IFGT=15645POKE15644,172:GOTO500
480 POKEGT,140:POKEGB,131:GOTO479
481 E=E+1:IFE>4THEN500
483 GOSUB830
490 GOTO300
495 REM -- MOVE FORWARD, TURN RIGHT OR TURN LEFT
500 M$=INKEY$:IFM$=""THEN500
510 E=0:IFM$="F"THEN550
520 IFM$="R"THEN620
530 IFM$="L"THEN660
540 GOTO500
545 REM ---- MOVE FORWARD
550 CLS:IF(X=A(100))*(D=4)THEN700
560 IF(X=A(101))*(D=2)THEN760
570 GOSUB870
580 IFVAL(MID$(B$,3,1))=1THEN710
590 Q=Q+1:X=X+Y
600 GOSUB830
610 GOTO300
615 REM ---- TURN RIGHT
620 CLS:D=D+1:IFD<5THEN640
630 D=1
640 GOSUB830
650 GOTO300
655 REM ---- TURN LEFT
660 CLS:D=D-1:IFD>0THEN680
670 D=4
680 GOSUB830
690 GOTO300
695 REM -- WIN OR LOSE
700 PRINT@338,"YOU LOSE.  OUT AT ENTRANCE.":GOTO770
710 FORI=10TO19:FORJ=3TO4:SET(I,J+Z):NEXTJ:Z=Z+1:NEXTI
720 FORI=13TO8STEP-1:SET(19,I):NEXTI:Z=0
730 FORI=19TO33:FORJ=8TO9:SET(I,J+Z):NEXTJ:Z=Z+1:NEXTI
740 PRINT@530,"ZZZAAPPPP!!"
750 PRINT@653,"YOU JUST RAN INTO THE ELECTRIFIED WALL!"
752 GOTO770
760 PRINT@333,"YOU WIN IN";Q;"MOVES. ";A(102);
762 PRINT"IS MINIMUM SCORE."
770 PRINT:PRINT
772 PRINTTAB(13)"DO YOU WANT TO TRY AGAIN?  (Y=YES, N=NO)"
780 M$=INKEY$:IFM$=""THEN780
790 IFM$="Y"RESTORE:GOTO80
800 IFM$="N"THEN820
810 GOTO780
820 CLS:PRINT@320,"OK.  COME BACK WHEN YOU'RE":END
825 REM -- TEST FOR DIRECTION YOU ARE FACING
830 IFD=1THENY=-10
840 IFD=2THENY=1
850 IFD=3THENY=10
860 IFD=4THENY=-1
865 REM -- FETCH ARRAY A CONTENTS FOR CURRENT LOCATION
870 B$=STR$(A(X+(Y*E)))
875 REM -- ROTATE ARRAY A CONTENTS FOR CURRENT DIRECTION
880 IFD=1THEN902
890 FORI=2TOD
892 B4=VAL(MID$(B$,4,1)):B5=VAL(MID$(B$,5,1))
894 B6=VAL(MID$(B$,6,1)):B3=VAL(MID$(B$,3,1))
896 P=10000+B4*1000+B5*100+B6*10+B3
```

```
900 B$=STR$(P):NEXTI
902 RETURN
905 REM -- ARRAY D DATA
910 DATA1,8,7,14,2,8,8,20,3,7,10,23,4,10
912 DATA8,29,5,10,10,38,6,10,10,45
925 REM -- ARRAY A DATA FOR MAZE NR 1
930 DATA0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
940 DATA0,0,0,0,0,0,0,0,0,0
942 DATA11001,11010,11100,11011,11100,11011,11000,11100,0,0
950 DATA10011,11100,10111,11001,10010,11110,10101,10101,0,0
952 DATA11101,10101,11011,10100,11101,11001,10110,10001,0,0
960 DATA10000,10000,11100,10011,10100,10101,11011,10110,0,0
962 DATA10111,10101,10011,11000,10000,10010,11010,11100,0,0
970 DATA11001,10010,11100,10101,10011,11100,11101,10101,0,0
972 DATA10011,11010,10110,10011,11110,10011,10010,10110,0,0
980 DATA60,57,14,8,7
985 REM -- ARRAY A DATA FOR MAZE NR 2
990 DATA0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
1000 DATA11101,11011,11000,11010,11000,11010,11100,11101,0,0
1002 DATA10011,11010,10000,11110,10011,11010,10100,10101,0,0
1010 DATA11001,11100,10001,11100,11001,11110,10001,10100,0,0
1012 DATA10101,10101,10111,10101,10101,11001,10110,10111,0,0
1020 DATA10000,10000,11010,10000,10100,10001,11010,11100,0,0
1022 DATA10101,10111,11101,10101,10101,10011,11100,10011,0,0
1030 DATA10001,11110,10001,10100,10011,11110,10101,11101,0,0
1032 DATA10011,11010,10110,10011,11010,11110,10011,10110,0,0
1040 DATA60,77,20,8,8
1045 REM -- ARRAY A DATA FOR MAZE NR 3
1050 DATA11001,11010,11100,11011,11000,11000,11100,0,0,0
1052 DATA10000,11100,10111,11101,10101,10101,10111,0,0,0
1060 DATA10101,10011,11000,10000,10110,10101,11101,0,0,0
1062 DATA10011,11100,10111,10111,11001,10010,10100,0,0,0
1070 DATA11101,10001,11000,11100,10101,11011,10110,0,0,0
1072 DATA10011,10110,10101,10111,10001,11110,11101,0,0,0
1080 DATA11001,11000,10110,11001,10010,11110,10101,0,0,0
1082 DATA10101,10101,11101,10101,11011,11000,10110,0,0,0
1090 DATA10001,10010,10100,10001,11010,10100,11001,0,0,0
1092 DATA10111,11011,10110,10011,11110,10011,10110,0,0,0
1100 DATA10,86,23,7,10
1105 REM -- ARRAY A DATA FOR MAZE NR 4
1110 DATA11011,11010,11000,11100,11001,11010,11000
1112 DATA11110,11011,11100,11001,11110,10101,10111
1114 DATA10011,11100,10101,11011,11100,10101
1120 DATA10101,11101,10011,11000,11010,10110,10101
1122 DATA11011,10100,10101,10100,10011,11000,10110
1124 DATA11001,11110,10001,11100,10001,10110
1130 DATA10001,11100,10001,11010,10100,11001,10100
1132 DATA10111,10001,11100,10101,10111,10101,11011
1134 DATA10110,10101,10101,11001,10110,10011
1140 DATA10001,11010,10110,11101,11101,10101,10001
1142 DATA10110,11101,11101,10011,11010,11010,10110
1144 DATA10011,10110,10011,11010,10010,10110
1145 DATA0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
1150 DATA30,59,29,10,8
1155 REM -- ARRAY A DATA FOR MAZE NR 5
1160 DATA11001,11010,11100,11011,11100,11001,11110
1162 DATA11011,11100,11101,10100,11101,10001,11000
1164 DATA10110,10101,11001,11000,10010,10100
1170 DATA10101,10011,10110,10101,11101,10011,10100
1172 DATA10111,11101,10001,10001,11010,11000,10010
1174 DATA10110,11101,10011,11000,10110,10101
1180 DATA10011,11110,10101,11001,11010,10100,11101
1182 DATA10101,11011,10110,11001,11010,10100,10111
1184 DATA11011,10000,10110,10101,11001,11100
1190 DATA10101,11011,10110,11101,11101,10101,11011
1192 DATA10010,10100,10111,10001,11010,11000,10110
1194 DATA10011,10000,11000,11110,10001,11110
1200 DATA10101,11101,10101,11011,11100,10101,10101
1202 DATA11001,10000,11110,10011,10110,10011,11010
1204 DATA10010,10110,10011,10110,10011,11110
```

Program continued

```
1210 DATA10,29,38,10,10
1215 REM -- ARRAY A DATA FOR MAZE NR 6
1220 DATA11001,11000,11010,11100,11001,11100,11011
1222 DATA11100,11011,11100,10111,10101,11101,10101
1224 DATA10111,10101,11001,10000,11010,10100
1230 DATA11000,10010,10110,10001,11000,10110,10101
1232 DATA10011,11110,10101,10101,11011,11000,10110
1234 DATA10011,11110,10011,11100,11001,10100
1240 DATA10011,11110,10101,11011,11000,11110,11101
1242 DATA10101,10111,10101,11001,11010,10000,11110
1244 DATA10101,11101,10011,10100,11101,10101
1250 DATA10011,11110,10101,11001,10100,10011,11010
1252 DATA10000,10110,10101,11001,11010,10100,10111
1254 DATA10001,11010,11010,10100,11001,10110
1260 DATA10011,11100,10011,11110,10101,11101,11101
1262 DATA10101,10011,11010,11011,10010,11010,11010
1264 DATA10010,10010,10110,10011,11010,11110
1270 DATA20,89,45,10,10
1280 END
```

Micro-Basketball

by Charles Weindorf

I stepped onto the court to play another game of "death" basketball. Death means no rules, no referees, and no sanity. I was at a disadvantage because my 130-pound, five-foot-seven-inch body didn't send the opposing six foot-seven inch center into a tailspin. I then asked, "Why kill myself? I like basketball, but not terminal injuries." That's how my search for a good basketball game program began. Finally, I decided to design one myself.

The Program

The Micro-Basketball program is designed for a Level II 16K TRS-80 and uses about 15.5K of memory. It should be input via cassette. For those willing to suffer a little keyboard cramp, the following hints should aid in the typing. In a line such as 4330, there are 15 spaces between two words. Counting all those spaces slows the typing process. Before typing the instructions (and cursing at me) here is a simple way to avoid the ordeal. First, leave out lines 3360-4550; then change lines 3360, 3400, and 3750 to CLS: RETURN. The program still offers you directions and a scorebook, but it only redraws the court.

Instructions

Most computer games are a battle of wits between human and computer, but this game's instructions make the reader wonder if his ammunition is running low. Some of the instructions for this game are complex. My friends who have played the game tell me that it is best to read the instructions quickly, and then refer to them as needed. With this advice in mind, I wrote the program with an option that allows you to refer to the instructions before any offensive or defensive play. If you choose to leave the instructions out, however, refer to this article to explain the procedures of the game.

The instructions are split into two parts: the directions (the total set of instructions) and the scorebook.

Directions

The directions explain the graphics and the game's limits. Included are the offensive and defensive courts, and the set of players assigned to each team. Your team (visitors) always has thin players, while the computer's team has fat ones. Each game is limited to two 15-minute halves (unless the game is tied and goes into overtime). Each offensive play uses up 20 seconds on the clock.

Scorebook

The scorebook section explains strategy and its effect on the shooting percentage. The three factors that change the shooting percentage are: team setup, team status, and a random number. Team setup determines the major portion of the shooting percentage (starts at 50 percent each play). Table 1 displays the three offensive and three defensive choices. Each offense has a different probability of success against a certain defense (just as in a real game). Table 2 gives all the possible offensive/defensive combinations and shows which are favorable to the offense or the defense. Any combination that is favorable to the offense increases the shooting percentage by 15 percent. Any combination favorable to the defense decreases the shooting percentage by 15 percent. Any other offensive/defensive combination increases the chance that a foul will occur from 15 to 35 percent.

Offensive Choices	Defensive Choices
outside shot (1)	3-2 defense (1)
inside shot (2)	2-1-2 defense (2)
choice shot (3)	man-man defense (3)

Table 1. *Choices*

Favored Offensive Setups

(1) off. vs. (2) def. (2) off. vs. (3) def. (3) off. vs. (1)def.

Favored Defensive Setups

(1) off. vs. (1) def. (2) off. vs. (2) def. (3) off. vs. (3) def.

Other Offensive/Defensive Setups

(1) off. vs. (3) def. (2) off. vs. (1) def. (3) off. vs. (2) def.

Table 2. *Setups*

Team status also changes the shooting percentage. Before each play, team status (aggressive or safe) is chosen along with team setup. To explain team status, I will use an example from a real basketball game. The defensive team has decided to play safely to stop any offensive plays that are designed to penetrate near the basket. The offensive team's play is aggressive (designed to penetrate), and the defense is able to stop the play before it is fully executed, forcing the shooter to take an uncomfortable shot. The shot is not

likely to go in. The next time down the court, the defense decides to play aggressively (to stop any shots away from the basket), but the offense decides to play aggressively also. This allows the offensive team to break through and give the shooter a comfortable shot. This shot probably goes in.

In Micro-Basketball, there are small arrows that signify aggressive and safe play. The arrows may seem confusing at first, but are very useful once they're understood. If the arrows for your team are pointed at the other team, your team plays aggressively; if the arrows are pointed away, they play safely. The effect upon the shooting percentage is this: If the offensive status and defensive status are the same, the shooting percentage increases by 10 percent. If the offensive and defensive status differ, shooting percentage decreases by 10 percent.

A random number is the third factor in shooting percentage. This random number from one to ten is added to the shooting percentage to make the game a higher scoring contest.

For example, the offense is playing an inside shot while the defense is playing the three-two. In Table 2, you find that it is one of the setups that has no effect on the shooting percentage. Both offense and defense are playing aggressively, so the shooting percentage increases 10 percent, and there

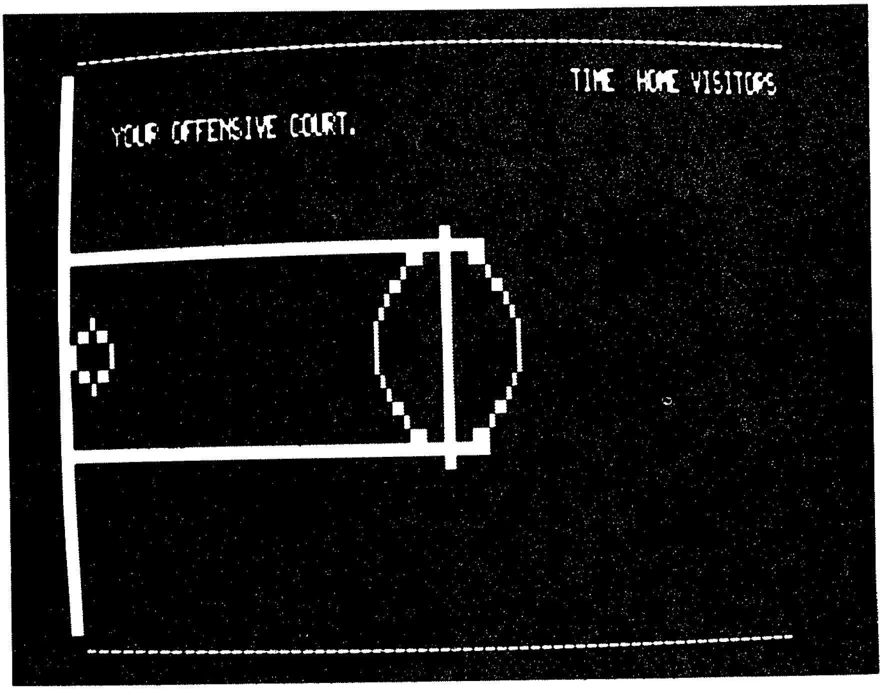


Photo 1. *The player's offensive half court. Character string combinations are used to draw the basket and foul circle.*

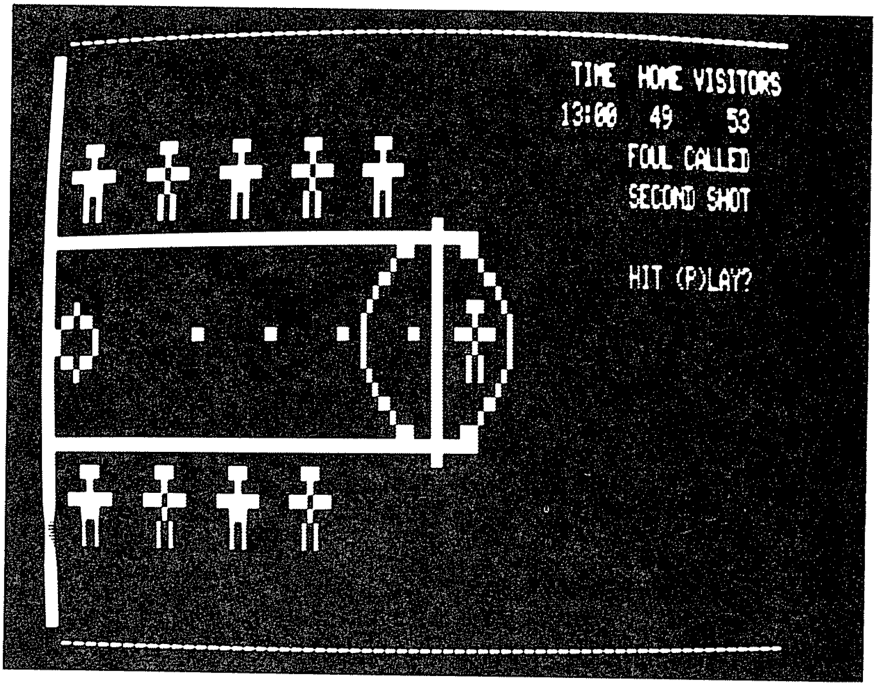


Photo 2. *The two teams lined up for the second foul shot.*

is a random number of three. Since the shooting percentage starts at 50, the final shooting percentage equals 50 + 0 + 10 + 3 = 63%.

Subroutines

The fun part of Micro-Basketball is its graphics, particularly seeing your men pass, dribble, and shoot. (At least it's fun while you're winning.) Even the computerized cheering sections get into the action. For organization's sake, I separated the program into 11 major subroutines, each designed to handle a specific part of the graphics. Dribbling (1720), shooting (2770), fouls (2910), player positioning (1500), and court drawing (1030) are called from the main body of the program (0-1020). The combination of these subroutines simulates the deployment of offenses and defenses in an actual basketball game.

All of the graphics are done using PRINT@ statements. The position of an object is found by each individual subroutine from a set of arrays using tables of numbers as starting points. From these, the computer draws the figures. The arrays C (court design), OF (offensive players), DF (defensive players), and PM (the moving basketball) are in two subdivisions, one for

each half-court. The use of the subroutines and the arrays permits the program to execute the graphics quickly, enhancing the realism of the game.

The subroutine that prevents the basketball from erasing the players and the court could be useful in many programs. The @-to-POINT subroutine (lines 3210-3290) converts a PRINT@ number (MO) and changes it into the number of lines down (MP) and the number of spaces over (MQ). It then checks all the blocks within a three-by-two space by using MQ and MP to determine the POINT coordinates. The formula for the x-coordinate is MQ * 2 + A7; the y-coordinate is MP * 3 + A6 (where A7 and A6 are the parameters of the loops). If there are any blocks SET within the space, the subroutine returns a 1 in MR. The shooting subroutine, in turn, skips printing the ball at that space.

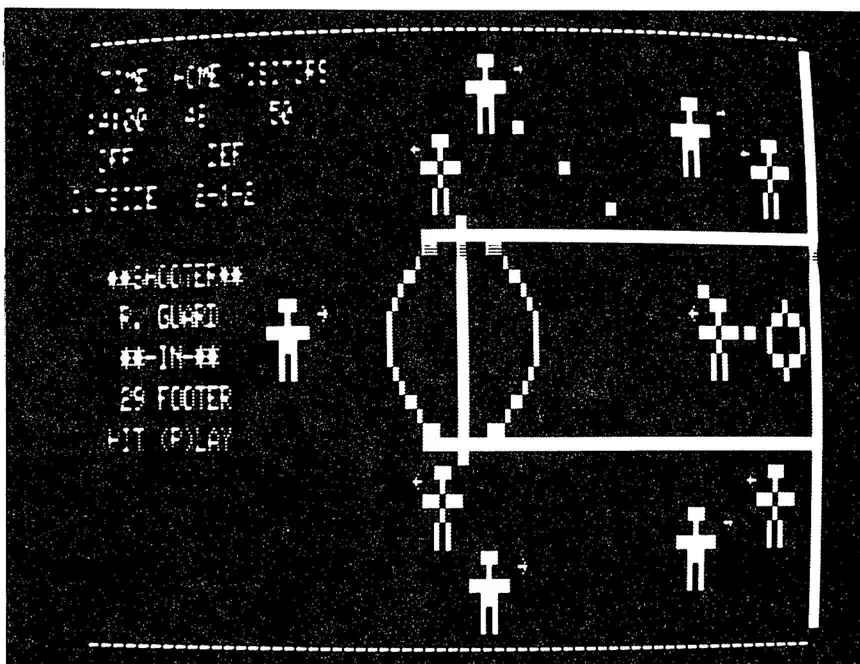


Photo 3. *Action on the players' defensive court prior to the shot. The scoreboard to the left gives the facts of the play. (Photographs by Thomas Cwalina)*

Sequence of Play

Micro-Basketball proceeds much like a regular basketball game. Each offensive play has dribbling, passing, and shooting, as well as a possibility of fouls, blocks, three-pointers, and even slamdunks. There are three major segments that make up each offensive play. The setup, the action, and the transition sections simulate most of the play of a regular game.

In the setup section, the player and the computer pick offensive and defensive strategies. The offensive player picks the type of shot, a shooter, and a status, while the defensive player chooses a defense and a status. The two teams are then placed on the court. Since there are three types of defense, the defensive positioning will vary, but the offense will always be placed on the court as shown in Table 3. The scoreboard (the time clock and space directly beneath it) is then set up. It shows the offensive/defensive choices and the shooter. The play is ready to begin.

Right guard (G)		G	
Right foreward (R)	R		
Point man (P)			P
Left foreward (L)	L		
Center (C)		C	

Table 3. *Positions*

The action sequence continues until the offensive play is completed. As play begins, either a foul will be called, or the ball will be given to the shooter. If there is a foul called, the men line up as in a real game, and two shots are taken. If play continues, the ball is dribbled and passed by the point man to the shooter.

In the next action sequence, the shooter will either take the shot or have it blocked. If the ball is blocked, play goes to the transition section. The shot, once away from the shooter, will either go in or be missed. If the shot goes in, there is a chance that it will be a slamdunk or a three-pointer. An inside shot of five feet or less, or any inside or choice shot by the center, is counted as a slamdunk. A shot of 27 feet or greater is counted as a three-pointer. If the shot misses, there is a 40 percent chance of an offensive rebound. An offensive rebound sends the program back to the setup section. If a shot is made or missed, the program goes to the transition section.

The transition section handles the post-action play. If there is a blocked shot, slamdunk, or three-pointer in the action section, a cheering section (one for each team) jumps up and down for the good play. After this is done, the other team is given control of the ball, and the program goes back to the setup section.

Well, armchair basketball fans, our time has come. Micro-Basketball lets us dribble, pass, shoot, slamdunk, block, and cheer to our heart's content.

Program Listing. *Micro-Basketball*

```
  0 REM     CHARLES E. WEINDORF     MICRO BASKETBALL      1/80
 10 CLEAR 200:
    CLS
 20 DIM DF(2,3,5),OF(2,5),C(2,6):
    GOTO 4620
 30 GOSUB 3330
 40 T = 15:
    S = 00
 50 TE = 1:
    GOSUB 1030
 60 PB = 170
 70 IF T < 10
     THEN
      PA = 171:
      GOTO 90
 80 PA = 170
 90 GOSUB 3300
100 PRINT @242,"WHAT PLAY?";:
    PRINT @306,"(1)OUTSIDE";
110 PRINT @370,"(2)INSIDE";:
    PRINT @434,"(3)CHOICE";
120 PRINT @498,"(4)SCOREBOOK";:
    PRINT @562,"(5)DIRECTIONS";
130 K$ = INKEY$:
    IF K$ = ""
     THEN
      130
140 PL = VAL(K$):
    IF PL < = 0 OR PL > 5
     THEN
      130
150 FOR A9 = 242 TO 562 STEP 64:
     PRINT @A9,"              ";:
     NEXT
160 PRINT @242,"WHAT MAN?";
170 K$ = INKEY$:
    ON PL GOTO 180,230,280,340,350
180 PRINT @306,"(1)POINT MAN";
190 PRINT @370,"(4)CENTER";:
    PRINT @434,"(5)R. GUARD";
200 K$ = INKEY$:
    IF K$ = ""
     THEN
      200
210 ML = VAL(K$):
    IF ML > < 1 AND ML > < 4 AND ML > < 5
     THEN
      200
220 GOTO 360
230 PRINT @306,"(2)R. FORWARD";:
    PRINT @370,"(3)L. FORWARD";
240 PRINT @434,"(4)CENTER";
250 K$ = INKEY$:
    IF K$ = ""
     THEN
      250
260 ML = VAL(K$):
    IF ML > < 2 AND ML > < 3 AND ML > < 4
     THEN
      250
270 GOTO 360
280 PRINT @306,"(1)POINT MAN";:
    PRINT @370,"(2)R. FORWARD";
290 PRINT @434,"(3)L. FORWARD";:
    PRINT @498,"(4)CENTER";
300 PRINT @562,"(5)R. GUARD";
310 K$ = INKEY$:
```

(Note: This listing is formatted for readability. Memory space is tight, and the user is advised to compress where possible.)

```
    IF K$ = ""
     THEN
      310
320 ML = VAL(K$):
    IF ML < = Ø OR ML > 5
     THEN
      31Ø
33Ø GOTO 36Ø
34Ø GOSUB 375Ø:
    GOTO 5Ø
35Ø GOSUB 34ØØ:
    GOTO 5Ø
36Ø K$ = INKEY$:
    FOR A9 = 242 TO 882 STEP 64:
     PRINT @A9,"               ";:
     NEXT
37Ø PRINT @242,"STATUS";:
    PRINT @3Ø6,"(1)AGGRESSIVE";:
    PRINT @37Ø,"(2)SAFE";
38Ø K$ = INKEY$:
    IF K$ = ""
     THEN
      38Ø
39Ø SL = VAL(K$):
    IF SL > < 1 AND SL > < 2
     THEN
      38Ø
4ØØ K$ = INKEY$
41Ø FOR A9 = 242 TO 37Ø STEP 64:
     PRINT @A9,"               ";:
     NEXT
42Ø REM  *** COMP DEF. CHOICE
43Ø PC = RND(3):
    SC = RND(2)
44Ø PA$ = P1$:
    PB$ = P3$:
    PC$ = P2$:
    PD$ = P4$
45Ø K$ = INKEY$
46Ø GOSUB 151Ø
47Ø FW = 194:
    FX = 218:
    FY = 7Ø6:
    FZ = 718:
    KK = 1:
    TA = 241:
    GOSUB 187Ø
48Ø IF TM = 1
     THEN
      TM = Ø:
      GOTO 57Ø
49Ø GOSUB 278Ø
5ØØ IF ML = 1
     THEN
      53Ø
51Ø GOSUB 173Ø
52Ø IF BK = 1
     THEN
      GOSUB 228Ø:
      BK = Ø:
      GOTO 55Ø
53Ø GOSUB 284Ø
54Ø GOSUB 239Ø
55Ø PRINT @TA + 45Ø,"HIT (P)LAY";
56Ø K$ = INKEY$:
    IF K$ < > "P"
     THEN
      56Ø
57Ø GOSUB 126Ø
58Ø K$ = INKEY$
```

```
590 TE = 2:
    GOSUB 1030
600 PB = 128
610 IF T < 10
     THEN
      PA = 129:
      GOTO 630
620 PA = 128
630 GOSUB 3300
640 PRINT @192,"WHAT DEFENSE?";:
    PRINT @256,"(1)3-2";:
    PRINT @320,"(2)2-1-2";
650 PRINT @384,"(3)MAN-MAN";:
    PRINT @448,"(4)SCOREBOOK";
660 PRINT @512,"(5)DIRECTIONS";
670 K$ = INKEY$: •
    IF K$ = ""
     THEN
      670
680 PC = VAL(K$):
    IF PC < = 0 OR PC > 5
     THEN
      670
690 ON PC GOTO 720,720,720,700,710
700 GOSUB 3750:
    GOTO 590
710 GOSUB 3400:
    GOTO 590
720 K$ = INKEY$:
    FOR A9 = 192 TO 512 STEP 64:
     PRINT @A9,"                ";:
     NEXT
730 PRINT @192,"STATUS";:
    PRINT @256,"(1)AGGRESSIVE";:
    PRINT @320,"(2)SAFE";
740 K$ = INKEY$:
    IF K$ = ""
     THEN
      740
750 SC = VAL(K$):
    IF SC > < 1 AND SC > < 2
     THEN
      740
760 FOR A9 = 192 TO 320 STEP 64:
     PRINT @A9,"                ";:
     NEXT
770 REM  ** COMP OFF. CHOICE
780 PL = RND(3):
    SL = RND(2)
790 ON PL GOTO 800,840,860
800 XX = RND(3):
    IF XX = 1
     THEN
      ML = 1:
      GOTO 880
810 IF XX = 2
     THEN
      ML = 4:
      GOTO 880
820 ML = 5
830 GOTO 880
840 ML = RND(3) + 1
850 GOTO 880
860 ML = RND(5)
870 GOTO 880
880 K$ = INKEY$
890 PA$ = P2$:
    PB$ = P4$:
    PC$ = P1$:
    PD$ = P3$
900 GOSUB 1510
```

```
910 FW = 226:
    FX = 250:
    FY = 750:
    FZ = 762:
    KK = - 1:
    TA = 193:
    GOSUB 1870
920 IF TM = 1
     THEN
      TM = 0:
      GOTO 1010
930 GOSUB 2780
940 IF ML = 1
     THEN
      970
950 GOSUB 1730
960 IF BK = 1
     THEN
      GOSUB 2280:
      BK = 0:
      GOTO 990
970 GOSUB 2840
980 GOSUB 2390
990 PRINT @TA + 450,"HIT (P)LAY";
1000 K$ = INKEY$:
     IF K$ < > "P"
      THEN
       1000
1010 GOSUB 1260
1020 K$ = INKEY$:
     GOTO 50
1030 CLS :
     PRINT @C(TE,6), STRING$(32, CHR$(140));
1040 PRINT @C(TE,6) + 320, STRING$(32, CHR$(140));
1050 PRINT @1, STRING$(63,"-");
1060 PRINT @961, STRING$(62,"-");
1070 IF TE = 1
      THEN
       PRINT @0, CHR$(176);:
       GOTO 1090
1080 PRINT @63, CHR$(176);
1090 FOR A9 = C(TE,1) TO C(TE,1) + 895 STEP 64:
      PRINT @A9,B$;:
      NEXT
1100 FOR A9 = C(TE,2) TO C(TE,2) + 320 STEP 64:
      PRINT @A9,B$;:
      NEXT
1110 PRINT @C(TE,3), CHR$(140) + CHR$(172) + CHR$(188);
1120 PRINT @C(TE,3) + 67, CHR$(137) + CHR$(164);:
     PRINT @C(TE,3) + 133, CHR$(169);
1130 PRINT @C(TE,3) + 197, CHR$(154);:
     PRINT @C(TE,3) + 259, CHR$(152) + CHR$(134);
1140 PRINT @C(TE,3) + 320, CHR$(140) + CHR$(142) + CHR$(143);
1150 PRINT @C(TE,3) - 4, CHR$(188) + CHR$(156) + CHR$(140);
1160 PRINT @C(TE,3) + 58, CHR$(152) + CHR$(134);:
     PRINT @C(TE,3) + 121, CHR$(150);
1170 PRINT @C(TE,3) + 185, CHR$(165);:
     PRINT @C(TE,3) + 250, CHR$(137) + CHR$(164);
1180 PRINT @C(TE,3) + 316, CHR$(143) + CHR$(141) + CHR$(140);
1190 GOSUB 1200:
     GOTO 1230
1200 PRINT @C(TE,4), CHR$(152) + CHR$(134) + CHR$(140) + CHR$(144);
1210 PRINT @C(TE,4) + 64, CHR$(137) + CHR$(164) + CHR$(140) +
     CHR$(129);
1220 RETURN
1230 PRINT @C(TE,5)," TIME  HOME VISITORS";
1240 RETURN
1250 REM  CLOCK COUNTER
1260 S = S - 20
1270 IF S < 00
```

```
     THEN
      S = 40:
      T = T - 1
1280 IF T = 0 AND S = 0
     THEN
      1300
1290 RETURN
1300 CLS :
     VV = VV + 1:
     IF VV = 2
      THEN
       1400
1310 TE = 1:
     GOSUB 1030
1320 PA = 171:
     PB = 170:
     GOSUB 3300:
     FOR A9 = 1 TO 3000:
      NEXT
1330 CLS
1340 PRINT CHR$(23):
     PRINT @472,"HALFTIME"
1350 FOR A9 = 1 TO 3000:
      NEXT
1360 CLS :
     PRINT "**** DO YOU WANT TO QUIT NOW AND SAVE FACE";:
     INPUT II$
1370 IF II$ = "NO"
      THEN
       T = 15:
       S = 0:
       GOTO 50
1380 IF II$ > < "YES"
      THEN
       1360
1390 PRINT "TRY AGAIN SOMETIME.":
     END
1400 IF H(1) < > H(2) GOTO 1430
1410 T = 4:
     S = 0:
     VV = 1:
     PRINT CHR$(23):
     PRINT @472,"OVERTIME!"
1420 FOR A9 = 1 TO 3000:
      NEXT :
     GOTO 50
1430 IF H(1) < H(2)
      THEN
       1480
1440 PRINT TAB(21);"****** YOU WIN ******"
1450 PRINT TAB(21);"***** PURE LUCK *****"
1460 PRINT TAB(21);"HOME";H(2);:
     PRINT TAB(31);"VISITORS";H(1);
1470 GOTO 1470
1480 PRINT TAB(21);"******* I WIN *******"
1490 GOTO 1460
1500 REM  ** SET UP PLAYERS ON THE SCREEN **
1510 FOR A9 = 1 TO 5
1520  PRINT @OF(TE,A9),PC$;:
      PRINT @OF(TE,A9) + 64,PD$;
1530  PRINT @DF(TE,PC,A9),PA$;:
      PRINT @DF(TE,PC,A9) + 64,PB$;
1540  NEXT
1550 ON TE GOTO 1560,1590
1560 IF SL = 1
      THEN
       OF$ = CHR$(93):
       YW = - 1:
       GOTO 1610
1570 OF$ = CHR$(94):
     YW = 4
```

```
1580 GOTO 1610
1590 IF SL = 1
      THEN
       OF$ = CHR$(94):
       YW = 4:
       GOTO 1610
1600 OF$ = CHR$(93):
     YW = - 1
1610 ON TE GOTO 1620,1650
1620 IF SC = 1
      THEN
       DF$ = CHR$(94):
       YX = 4:
       GOTO 1670
1630 DF$ = CHR$(93):
     YX = - 1
1640 GOTO 1670
1650 IF SC = 1
      THEN
       DF$ = CHR$(93):
       YX = - 1:
       GOTO 1670
1660 DF$ = CHR$(94):
     YX = 4
1670 FOR A9 = 1 TO 5
1680  PRINT @OF(TE,A9) + YW,OF$;
1690  PRINT @DF(TE,PC,A9) + YX,DF$;
1700  NEXT
1710 RETURN
1720 REM   ** DRIBBLE **
1730 IF TE = 1
      THEN
       A4 = 63:
      ELSE
       A4 = 68
1740 FOR A6 = 1 TO A5
1750  PRINT @OF(TE,ML) + A4, CHR$(131);:
      GOSUB 1810
1760  PRINT @OF(TE,ML) + A4, CHR$(140);:
      GOSUB 1810
1770  PRINT @OF(TE,ML) + A4, CHR$(176);:
      GOSUB 1810
1780  PRINT @OF(TE,ML) + A4, CHR$(140);:
      GOSUB 1810
1790  PRINT @OF(TE,ML) + A4, CHR$(131);:
      GOSUB 1810
1800  GOTO 1830
1810  FOR A7 = 1 TO 20:
       NEXT
1820  RETURN
1830  NEXT A6
1840 PRINT @OF(TE,ML) + A4, CHR$(128);
1850 RETURN
1860 REM   **LOGIC**
1870 ON PL GOTO 1880,1900,1920
1880 PRINT @TA + 63,"OUTSIDE";
1890 GOTO 1930
1900 PRINT @TA + 64,"INSIDE";
1910 GOTO 1930
1920 PRINT @TA + 64,"CHOICE";
1930 PRINT @TA + 1,"OFF      DEF";
1940 ON PC GOTO 1950,1970,1990
1950 PRINT @TA + 72,"  3-2";
1960 GOTO 2000
1970 PRINT @TA + 72," 2-1-2";
1980 GOTO 2000
1990 PRINT @TA + 72,"MAN-MAN";
2000 PRINT @TA + 194,"**SHOOTER**";
2010 ON ML GOTO 2020,2040,2060,2080,2100
2020 PRINT @TA + 259,"POINT MAN";
2030 GOTO 2110
```

```
2040 PRINT @TA + 258,"R. FORWARD";
2050 GOTO 2110
2060 PRINT @TA + 258,"L. FORWARD";
2070 GOTO 2110
2080 PRINT @TA + 260,"CENTER";
2090 GOTO 2110
2100 PRINT @TA + 259,"R. GUARD";
2110 S% = 50:
     IF PL = PC
      THEN
       2140
2120 IF PC = PL + 1 OR PL = PC + 2
      THEN
       2130:
      ELSE
       GOTO 2160
2130 S% = S% + 15:
     GOTO 2170
2140 S% = S% - 15
2150 GOTO 2170
2160 F% = 20
2170 IF SL = SC
      THEN
       S% = S% + 10:
      ELSE
       S% = S% - 10
2180 PRINT @TA + 322,"HIT (P)LAY?";
2190 K$ = INKEY$:
     IF K$ > < "P"
      THEN
       2190
2200 FF = 15:
     FF = FF + F%:
     IF RND(100) < FF
      THEN
       2920
2210 F% = 0
2220 S% = S% + RND(10)
2230 PRINT @TA + 322," SHOT%="S%;
2240 IF RND(100) < S%
      THEN
       MM = 1:
       H(TE) = H(TE) + 2:
       RETURN
2250 MM = 0
2260 IF RND(100) > 35
      THEN
       RETURN :
      ELSE
       BK = 1
2270 RETURN
2280 FOR A9 = 1 TO 10:
      PRINT @TA + 323,"BLOCKED! ";
2290  FOR A8 = 1 TO 50:
       NEXT :
      PRINT @TA + 323,"         ";
2300  FOR A8 = 1 TO 50:
       NEXT :
      NEXT
2310 GOSUB 4820:
     RETURN
2320 RETURN
2330 IF RND(100) > 40
      THEN
       RETURN
2340 PRINT @TA + 321,"OFF REBOUND!";
2350 PRINT @TA + 386,"HIT (P)LAY";
2360 K$ = INKEY$:
     IF K$ < > "P"
      THEN
       2360
```

```
2370 IF TE = 1
      THEN
       50:
      ELSE
       GOTO 590
2380 RETURN
2390 REM  **-IN-**
2400 IF MM = 0
      THEN
       GOTO 2740:
      ELSE
       MM = 0
2410 PRINT @TA + 323,"**-IN-** ";
2420 GOSUB 2430:
     GOTO 2490
2430 GOSUB 3300
2440 FOR A9 = 1 TO 5
2450  PRINT @C(TE,4),"    ";:
      PRINT @C(TE,4) + 64,"    ";
2460  GOSUB 1200
2470  NEXT
2480 RETURN
2490 IF PL = 1
      THEN
       HD = 20 + RND(10):
       GOTO 2510
2500 GOTO 2550
2510 IF HD < 27
      THEN
       PRINT @TA + 386,HD;"FOOTER";:
       RETURN
2520 H(TE) = H(TE) + 1:
     GOSUB 3300:
     FOR A9 = 1 TO 10:
      PRINT @TA + 386,"3 POINTER";
2530  FOR A8 = 1 TO 50:
       NEXT :
      PRINT @TA + 386,"          ";:
      FOR A8 = 1 TO 50:
       NEXT :
      NEXT
2540 GOSUB 4820:
     RETURN
2550 IF ML < > 4
      THEN
       2690
2560 FOR A9 = 1 TO 4
2570  PRINT @TA + 387,"S";:
      GOSUB 1810
2580  PRINT @TA + 388,"L";:
      GOSUB 1810
2590  PRINT @TA + 389,"A";:
      GOSUB 1810
2600  PRINT @TA + 390,"M";:
      GOSUB 1810
2610  PRINT @TA + 391,"D";:
      GOSUB 1810
2620  PRINT @TA + 392,"U";:
      GOSUB 1810
2630  PRINT @TA + 393,"N";:
      GOSUB 1810
2640  PRINT @TA + 394,"K";:
      GOSUB 1810
2650  PRINT @TA + 387,"        ";
2660  NEXT
2670 GOSUB 4820
2680 RETURN
2690 IF PL < > 2
      THEN
       2720
2700 SD = RND(15):
```

```
     IF SD < = 5 GOSUB 2560:
     RETURN
2710 PRINT @TA + 386,SD;"FOOTER";:
     RETURN
2720 SD = RND(25) + 5:
     IF SD < 27
      THEN
       2710:
      ELSE
       2520
2730 RETURN
2740 PRINT @TA + 323,"* MISS * ";:
     FOR A9 = 1 TO 500:
      NEXT
2750 GOSUB 2330
2760 RETURN
2770 REM  ** SHOOT **
2780 QQ = ML:
     ML = 1:
     GOSUB 1730
2790 ON QQ GOTO 2820,2800,2810,2820,2820
2800 ML = 5:
     GOSUB 1730:
     GOTO 2820
2810 ML = 4:
     GOSUB 1730:
     GOTO 2820
2820 ML = QQ
2830 RETURN
2840 FOR AA = OF(TE,ML) TO C(TE,4) STEP PM(TE,ML)
2850  MO = AA
2860  GOSUB 3210
2870  IF MR = 1
       THEN
        2890
2880  PRINT @AA, CHR$(176);
2890  NEXT
2900 RETURN
2910 REM  *** FOUL SHOTS ***
2920 IF TE = 1
      THEN
       OF(TE,1) = OF(TE,1) - 15:
      ELSE
       OF(TE,1) = OF(TE,1) + 13
2930 FOR D9 = 1 TO 2:
      GOSUB 1030
2940  GOSUB 3300
2950  PRINT @TA,"FOUL CALLED";
2960  IF D9 = 1
       THEN
        PRINT @TA + 64,"FIRST SHOT ";:
       ELSE
        PRINT @TA + 64,"SECOND SHOT";
2970  FOR A9 = FW TO FX STEP 12
2980   PRINT @A9,PA$;:
       PRINT @A9 + 64,PB$;
2990   NEXT
3000  FOR A9 = FY TO FZ STEP 12
3010   PRINT @A9,PA$;:
       PRINT @A9 + 64,PB$;
3020   NEXT
3030  FOR A9 = FW + 6 TO FX - 6 STEP 12
3040   PRINT @A9,PC$;:
       PRINT @A9 + 64,PD$;
3050   NEXT
3060  FOR A9 = FY + 6 * KK TO FZ + 6 * KK STEP 12
3070   PRINT @A9,PC$;:
       PRINT @A9 + 64,PD$;
3080   NEXT
3090  IF TE = 1
       THEN
```

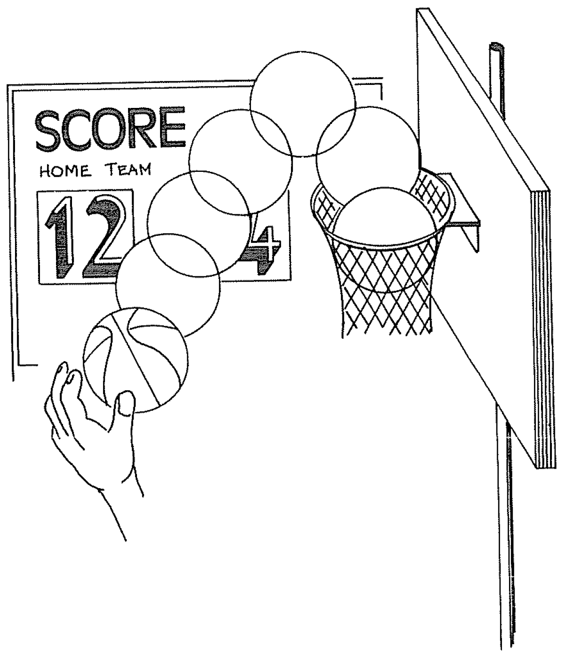
```
 DC = 482:
ELSE
 DC = 476
3100  PRINT @DC,PC$;:
      PRINT @DC + 64,PD$;
3110  PRINT @TA + 192,"HIT (P)LAY?";
3120  K$ = INKEY$:
      IF K$ > < "P"
       THEN
        3120
3130  ML = 1:
      GOSUB 2840
3140  IF RND(100) < 80
       THEN
        GOSUB 2430:
        H(TE) = H(TE) + 1
3150  GOSUB 3300
3160  NEXT
3170 IF TE = 1
      THEN
       OF(TE,1) = OF(TE,1) + 15:
      ELSE
       OF(TE,1) = OF(TE,1) - 13
3180 FOR RR = 1 TO 700:
      NEXT
3190 TM = 1
3200 F% = 0:
     RETURN
3210 REM  ** 'SET' CHECK **
3220 MP = INT(MO / 64)
3230 MQ = MO - (MP * 64)
3240 FOR A7 = 0 TO 1
3250  FOR A6 = 0 TO 2
3260   IF POINT(MQ * 2 + A7,MP * 3 + A6)
        THEN
         MR = 1:
         GOTO 3290
3270   NEXT :
      NEXT
3280 MR = 0
3290 RETURN
3300 PRINT @PA,T;:
     PRINT @PB + 3,S;:
     PRINT @PB + 8,H(2);:
     PRINT @PB + 15,H(1);
3310 PRINT @PB + 3,":";:
     IF S = 0
      THEN
       PRINT @PB + 5,"0";
3320 RETURN
3330 PRINT CHR$(23)
3340 PRINT @462,"MICRO  BASKETBALL"
3350 FOR A9 = 1 TO 3000:
      NEXT
3360 CLS
3370 INPUT "***** WOULD YOU LIKE DIRECTIONS? (Y)ES OR (N)O";ZZ$
3380 IF ZZ$ = "N"
      THEN
       RETURN
3390 IF ZZ$ < > "Y"
      THEN
       CLS :
       GOTO 3370
3400 CLS :
     PRINT "**** OBJECT OF THE GAME: TO OUT-SCORE THE COMPUTER";
3410 PRINT " WITHIN THE                                ALLOTTED TIME BY";
3420 PRINT " CHOOSING THE CORRECT                                  COMBINA";
3430 PRINT "TION OF OFFENSIVE AND DEFENSIVE                                ";
3440 PRINT "PLAYS."
3450 PRINT
3460 PRINT "DESCRIPTION OF THE GAME: THE GAME IS PLAYED ON TWO HALF-"
```

```
      ;
3470 PRINT "COURTS,                                    ONE FOR YOUR OFFENSE AND
      ";
3480 PRINT " ONE FOR YOUR                                     DEFENSE. A SCORE
      ";
3490 PRINT "BOARD IS IN THE CORNER                                  OF THE ";
3500 PRINT " SCREEN, UNDER WHICH A PLAY BY
      ";
3510 PRINT "PLAY DESCRIPTION IS GIVEN."
3520 PRINT
3530 PRINT "*********** THE PLAYERS: YOUR PLAYERS:";P2$;
3540 PRINT "THE COMPUTER'S:";P1$;"."
3550 PRINT TAB(38);P4$;:
     PRINT TAB(57);P3$;
3560 PRINT
3570 PRINT @896,"ANY KEY TO CONTINUE?";
3580 K$ = INKEY$:
     IF K$ = ""
      THEN
       3580
3590 CLS
3600 PRINT "*************** LIMITS: THE GAME IS LIMITED TO TWO 15 ";
3610 PRINT "MINUTE                                   HALVES. EACH OFFENSIVE
      ";
3620 PRINT "PLAY EQUALS 20                                 SECONDS. THE ";
3630 PRINT "OFFENSIVE RESTRICTIONS ON                               EACH "
      ;
3640 PRINT "PLAYER ARE";
3650 PRINT " DESCRIBED IN THE                                      ";
3660 PRINT "'SCOREBOOK' SECTION."
3670 PRINT "ANY KEY?"
3680 K$ = INKEY$:
     IF K$ = ""
      THEN
       3680
3690 CLS :
     TE = 1:
     GOSUB 1030
3700 PRINT @132,"YOUR OFFENSIVE COURT.";
3710 FOR A9 = 1 TO 2500:
      NEXT
3720 CLS :
     TE = 2:
     GOSUB 1030
3730 PRINT @165,"YOUR DEFENSIVE COURT.";
3740 FOR A9 = 1 TO 2500:
      NEXT
3750 CLS
3760 PRINT CHR$(23):
     PRINT @468,"SCOREBOOK":
     FOR A9 = 1 TO 1000:
      NEXT
3770 CLS
3780 PRINT "THE PLAYERS ARE":
     PRINT "-----------------------------"
3790 PRINT "THE RIGHT GUARD (G)":
     PRINT "THE RIGHT FORWARD (R)"
3800 PRINT "THE POINT MAN (P)":
     PRINT "THE LEFT FORWARD (L)"
3810 PRINT "THE CENTER (C)"
3820 PRINT @168,"G";
3830 PRINT @222,"R";:
     PRINT @306,"P";:
     PRINT @350,"L";:
     PRINT @424,"C";
3840 PRINT @448,"*** THE SET-UP ON THE OFFENSIVE COURT IS ";
3850 PRINT "PORTRAYED TO THE RIGHT.";
3860 PRINT "HIT 'C' TO CONTINUE."
3870 K$ = INKEY$:
     IF K$ < > "C"
      THEN
       3870
```

```
3880 GOSUB 3890:
     GOTO 3940
3890 CLS :
     PRINT "OFFENSIVE CHOICES                    DEFENSIVE CHOICES"
3900 PRINT "OUTSIDE SHOT (1)                    3-2 DEFENSE      (1)"
3910 PRINT "INSIDE SHOT  (2)                    2-1-2 DEFENSE    (2)"
3920 PRINT "CHOICE SHOT  (3)                    MAN-MAN DEFENSE (3)"
3930 RETURN
3940 PRINT
3950 PRINT "**** EACH PLAYER HAS A SHOOTING % OF 50 AT THE START OF "
     ;
3960 PRINT "EACH         PLAY, BUT IT CAN BE AFFECTED BY THE ";
3970 PRINT "DEFENSIVE CHOICE."
3980 PRINT :
     PRINT TAB(15);"THE FAVORED OFFENSIVE SET-UPS."
3990 PRINT STRING$(63,"-")
4000 PRINT "(1)OFF VS (2)DEF ** (2)OFF VS (3)DEF ** (3)OFF VS (1)DEF"
4010 PRINT "**** THIS WILL CAUSE THE SHOOTER'S % TO INCREASE 15%."
4020 PRINT
4030 PRINT "ANY KEY?"
4040 K$ = INKEY$:
     IF K$ = ""
      THEN
       4040
4050 GOSUB 3890
4060 PRINT
4070 PRINT TAB(15);"THE FAVORED DEFENSIVE SET-UPS."
4080 PRINT STRING$(63,"-")
4090 PRINT "(1)OFF VS (1)DEF ** (2)OFF VS (2)DEF ** (3)OFF VS (3)DEF"
4100 PRINT "**** THIS WILL CAUSE THE SHOOTER'S % TO DECREASE 15%."
4110 PRINT :
     PRINT TAB(25);"THE OTHERS"
4120 PRINT STRING$(63,"-")
4130 PRINT "(1)OFF VS (3)DEF ** (2)OFF VS (1)DEF ** (3)OFF VS (2)DEF"
4140 PRINT "**** THIS WILL NOT CHANGE THE SHOOTER'S %, BUT IT ";
4150 PRINT "INCREASES THE      CHANCE HE WILL BE FOULED."
4160 PRINT "ANY KEY?";
4170 K$ = INKEY$:
     IF K$ = ""
      THEN
       4170
4180 CLS
4190 PRINT TAB(22);"SPECIAL OPTIONS."
4200 PRINT STRING$(63,"-")
4210 PRINT "**** BOTH THE OFFENSE AND DEFENSE ARE REQUIRED TO PLAY ";
4220 PRINT "EITHER        'AGGRESSIVE' OR 'SAFE'. THIS ALSO AFFECTS T
     HE ";
4230 PRINT "SHOOTING %.        IF BOTH THE DEFENSE AND OFFENSE PLAY "
     ;
4240 PRINT "THE SAME, THE              SHOOTING % INCREASES 10%. IF "
     ;
4250 PRINT "THEY PLAY DIFFERENT, ";
4260 PRINT "THE %        DECREASES 10%."
4270 PRINT
4280 PRINT "**** THE TYPE OF PLAY (AGGRES OR SAFE) IS SHOWN BY ";
4290 PRINT CHR$(93);" OR "; CHR$(94);"."
4300 PRINT :
     PRINT "**** EXAMPLE. IF AN ARROW (ON AN OFFENSIVE PLAYER)";
4310 PRINT " IS POINTED AT              THE BASKET, THE TEAM IS PLAY"
     ;
4320 PRINT "ING AGGRESSIVELY. IF             THE DEFENSE HAS ARROW
     S";
4330 PRINT " POINTED AT THE BASKET, THEY             ARE PLAYING";
4340 PRINT " SAFELY. THE ";
4350 PRINT "RESULT IS THAT THE % IS              DECREASED BY ";
4360 PRINT "10%  (SEE RULE ABOVE)."
4370 PRINT "ANY KEY?";
4380 K$ = INKEY$:
     IF K$ = ""
      THEN
```

```
      4380
4390 CLS
4400 PRINT "********** SHOT % RANGE: THE SHOT % OF A PLAYER IS ALSO ";
4410 PRINT "ASSISTED                          BY A RANDOM NUMBER LESS
     ";
4420 PRINT "THAN 10."
4430 PRINT
4440 PRINT "**** LIMITS ON PLAYERS: (1)OUTSIDE SHOT-POINT MAN, CENTER
     ,";
4450 PRINT " AND                                          R. GUARD."
4460 PRINT "                       (2)INSIDE SHOT -CENTER, L. ";
4470 PRINT "FORWARD, AND                                        R. ";
4480 PRINT "FORWARD."
4490 PRINT "                       (3)CHOICE SHOT -ALL PLAYERS."
4500 PRINT "ANY KEY?"
4510 K$ = INKEY$:
     IF K$ = ""
      THEN
       4510
4520 IF HE$ = "D" OR HE$ = "S"
      THEN
       4540
4530 RETURN
4540 GOSUB 1030
4550 RETURN
4560 DATA 64,352,353,449,107,320
4570 DATA 127,352,353,507,65,352
4580 DATA 493,138,778,860,93,464,179,819,865,97
4590 DATA 476,196,708,728,216,456,194,706,736,224,487,132,772,854,87
4600 DATA 482,248,760,741,230,501,250,762,733,221,470,185,825,871,103
4610 DATA -6,62,-66,-68,60,6,66,-62,-60,68
4620 P1$ = CHR$(176) + CHR$(187) + CHR$(177) + CHR$(144)
4630 P3$ = CHR$(32) + CHR$(151) + CHR$(149) + " "
4640 P4$ = CHR$(32) + CHR$(150) + CHR$(148) + " "
4650 B$ = CHR$(191)
4660 P2$ = CHR$(176) + CHR$(155) + CHR$(177) + CHR$(144)
4670 FOR A9 = 1 TO 2:
      FOR A8 = 1 TO 6
4680   READ C(A9.A8)
4690   NEXT :
      NEXT
4700 FOR A9 = 1 TO 2
4710  FOR A8 = 1 TO 5
4720   READ OF(A9,A8)
4730   NEXT :
      NEXT
4740 FOR A9 = 1 TO 2:
      FOR A8 = 1 TO 3:
       FOR A7 = 1 TO 5
4750    READ DF(A9,A8,A7)
4760    NEXT :
       NEXT :
      NEXT
4770 FOR A9 = 1 TO 2:
      FOR A8 = 1 TO 5
4780   READ PM(A9,A8)
4790   NEXT :
      NEXT
4800 A5 = 2
4810 GOTO 30
4820 REM  ***** CROWD CHEERING
4830 CLS
4840 PRINT CHR$(23)
4850 PRINT @456,"THE CROWD GOES ";
4860 XL = RND(3):
     ON XL GOTO 4870,4880,4890
4870 PRINT "WILD":
     GOTO 4900
4880 PRINT "CRAZY":
     GOTO 4900
4890 PRINT "INSANE":
```

```
     GOTO 4900
4900 FOR A9 = 1 TO 700:
      NEXT :
     CLS
4910 FOR A9 = 384 TO 896 STEP 256
4920  PRINT @A9, STRING$(64, CHR$(131));
4930  NEXT
4940 FOR A9 = 31 TO 991 STEP 64
4950  PRINT @A9, CHR$(191) + CHR$(191);
4960  NEXT
4970 IF BK = 0
      THEN
       5000
```




```
4980 IF TE = 1
      THEN
       TE = 2:
      ELSE
       TE = 1
4990 HO$ = PA$:
     PA$ = PC$:
     PC$ = HO$:
     HO$ = PB$:
     PB$ = PD$:
     PD$ = HO$
5000 IF TE = 1
      THEN
       CW = 257:
       CR = 291:
       GOTO 5020
5010 CW = 291:
     CR = 257
5020 FOR A9 = CR TO CR + 512 STEP 256
5030  PRINT @A9,"";
5040  FOR A8 = 1 TO 5:
       PRINT PA$"  ";:
       NEXT
```

```
5050  PRINT @A9 + 64,"";
5060  FOR A8 = 1 TO 5:
       PRINT PB$"  ";:
       NEXT
5070  NEXT
5080 FOR A9 = CW TO CW + 512 STEP 256
5090  PRINT @A9,"";
5100  FOR A8 = 1 TO 5:
       PRINT PC$"  ";:
       NEXT
5110  PRINT @A9 + 64,"";
5120  FOR A8 = 1 TO 5:
       PRINT PD$"  ";:
       NEXT
5130  NEXT
5140 FOR A9 = 1 TO 5
5150  FOR A8 = CW - 64 TO CW + 448 STEP 256
5160   FOR A7 = A8 TO A8 + 24 STEP 6
5170    IF CC = 1 GOTO 5210
5180    CC = 1:
        PRINT @A7,PC$;:
        PRINT @A7 + 64,PD$;:
        PRINT @A7 + 128,"    ";
5190    NEXT
5200   GOTO 5230
5210   CC = 2:
       PRINT @A7,"    ";:
       PRINT @A7 + 64,PC$;:
       PRINT @A7 + 128,PD$;
5220   NEXT
5230  NEXT :
     NEXT
5240 RETURN
```

GRAPHICS

Images

Images

by Buzz Gorsky K8BG

The program shown in the Program Listing generates a series of lines like the spokes of a wheel from a randomly chosen point on the screen. It will then draw another pattern, delay, clear the screen, and start again. Let's see how it's done.

Starting at line 100, the K loop goes from one to two to draw the two patterns. X1 and Y1 are chosen randomly as values up to 127 and 47, respectively, so that the pair (X1,Y1) refers to a random point on the display in the format used by SET statements. This point will be the center for the radiating line pattern. Then, in line 110, T runs from zero to 170 drawn in increments of 10. T represents the angle in degrees (in this case 10) at which each line drawn will radiate. Since each line will run through the center of the circle, we only have to let T go this far.

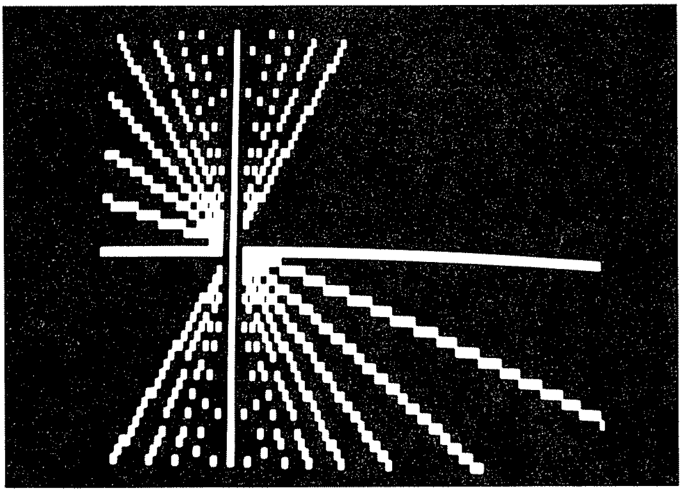


Figure 1. *Sample pattern of radial drawing*

When T is 90, a vertical line is needed. This is drawn by the FOR-NEXT loop involving L. In line 120, T1 is set equal to T times a constant to change the degree value to a radian value.

In line 130, X will run through the limits of the values which can be displayed. Y is then set equal to X times the tangent of T1. Y = X*TAN(V) is the equation for a line in a polar coordinate system.

In line 160, we check to see if the values of X2 and Y2 can be shown on the screen with a SET (X2,Y2) statement. If so, they are displayed at 170, and if

not, we go to 180. There we set X2 = X1 − X and Y2 = Y1 + Y. This then reflects the line just drawn through the center of the circle. If the values of X2 and Y2 can be displayed, then the SET statement displays them. Otherwise what happens depends on the value of Z.

If the first half of the line had terminated because values were off the screen, then Z would equal one. If this part of the line were also off limits, then we would reset Z equal to 0 and go to the next value of T. However, if Z were zero, then we would go to 190 and the next value of X. When X was completed, we'd have the next value of T. In this way, each half of the line is finished until it reaches the limits of the display.

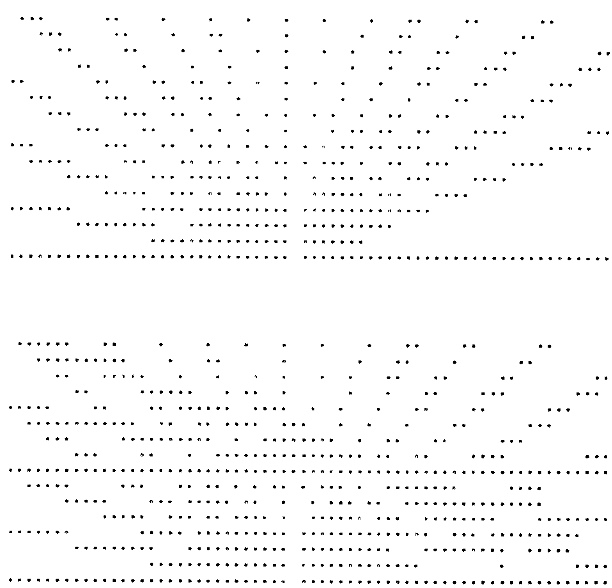


Figure 2. *Printout of radial drawings*

In line 200, there is a short wait and then the second pattern is drawn by going to the next value of K. A long delay follows, after which the program is run.

It's useless, I know, but fun to watch, and the radial line drawing technique might even find a place in something useful!

Program Listing

```
80 REM  RADIAL LINE DRAWING PROGRAM BY BUZZ GORSKY, K8BG
90 REM  THIS PROGRAM WILL BEGIN AT A RANDOM SPOT ON THE SCREEN AND
   DRAW A SERIES OF RADIAL LINES FROM THAT POINT. IT WIL REPEAT THE
    PROCESS TWO TIMES, HOLD THE DISPLAY, THEN BEGIN AGAIN.
100 RANDOM :
    CLS :
    FOR K = 1 TO 2:
     X1 = RND(127):
     Y1 = RND(47):
     REM  K SETS THE LIMIT OF 2 DISPLAYS BEFORE STARTING. X1 AND Y1
     ARE RANDOM DISPLACEMENTS FROM THE UPPER LEFT CORNER OF THE SCRE
     EN.
110  FOR T = 0 TO 170 STEP 10:
      IF T = 90
       THEN
        FOR L = 0 TO 47:
         SET(X1,L):
         NEXT L:
        NEXT T:
       REM  T IS RADIAL ANGLE IN DEGREES. FOR T=90 A VERTICAL LINE I
       S DRAWN RATHER THAN USING THE Y=X*TAN(T) EQUATION.
120  T1 = T * .0174533:
     REM  MODIFY T TO RADIANS
130  FOR X = 3 TO 127:
      REM  RUNS X THROUGH LIMITS OF DISPLAY
140   Y = X * TAN(T1):
      REM  SET Y ACCORDING TO RADIAL EQUATION OF STRAIGHT LINE.
150   X2 = INT(X + X1):
      Y2 = INT(Y + Y1):
      REM  MODIFY X AND Y ACCORDING TO RANDOM DISPLACEMENT
160   IF (X2 > 127 OR Y2 < 0 OR Y2 > 47)
       THEN
        Z = 1:
        GOTO 180:
        REM  IF X2 OR Y2 ARE OUT OF DISPLAY LIMITS THEN SET Z=1 AND
        GOTO 180 OTHERWISE DISPLAY
170   SET(X2,Y2)
180   X2 = INT(X1 - X):
      Y2 = INT(Y1 - Y):
      IF (X2 > - 1 AND X2 < 128 AND Y2 > - 1 AND Y2 < 48)
       THEN
        SET(X2,Y2)  :
       ELSE
        IF Z = 1
         THEN
          Z = 0:
          NEXT T:
         REM  CONTINUE THE RADIAL LINE IN A MIRROR IMAGE. IF X2 AND
        Y2 ARE OUT OF DISPLAY LIMITS AND Z=1 THEN NEXT ANGLE ELSE N
        EXT X.
190  Z = 0:
     NEXT :
    NEXT :
    REM  REST Z AND CONTINUE
200 FOR J = 1 TO 500:
     NEXT :
    NEXT :
    FOR J = 1 TO 30000:
     NEXT :
    RUN :
    REM  DELAY THEN DRAW NEXT PICTURE. AFTER 2 PIX HOLD, THEN START
    AGAIN.
```

HARDWARE

Regulate Your Video Monitor

CTR-80 Modifications

Regulate Your Video Monitor—for the TRS-80 Model I

by William Klungle

The Radio Shack TRS-80 is a lot of computer for the money invested. However, even with a good product such as the TRS-80, there is room for improvement. One of the areas that Radio Shack seems to have overlooked is the voltage regulation of the monitor. The regulation in the computer itself is excellent, but voltage regulation in the monitor is almost nonexistent. Any variation in the ac house current, such as may be caused by a pump or a dishwasher or a disk drive, results in a noticeable fluctuation of the video display.

Shortly after purchasing a TRS-80, I decided, for aesthetic reasons, to place the separate power module of the computer inside of the monitor case. This allowed the computer to reside on the family-room bookshelves and, with a small amount of rewiring, provided a single power switch for the entire system (see "Turn it Off!" *Microcomputing*, April '78, p. 114). As long as the monitor was on the workbench anyway, I took a close look at the power supply circuit to see what could be done about the regulation problem.

Regulating Transistor Circuit

The original circuit consisted of a half-wave rectifier and several RC filter networks (Figure 1). The characteristics of the transistor circuits tend to amplify even the small variations in supply voltage, so that without some type of regulation, the video display would never stand still.

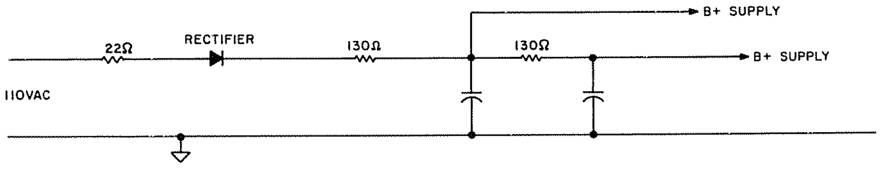


Figure 1. *Original circuit*

In the monitor's early life as a portable television, there were provisions made on the chassis for an additional transistor to be mounted. The chassis has been punched to mount a TO-66-style transistor in the same area where the rectifier is mounted. Voltage regulation can easily be added by using only four inexpensive parts. The regulator circuit is not critical in its specifications, and any components that meet or exceed the

minimum requirements may be used successfully. The original power supply provides approximately 120 V dc @ 350 mA. Any NPN silicon transistor in a TO-66-style case with a break-down voltage (VCEO) of over 150 volts and a minimum current rating (Ic) of 500 mA should work.

Unfortunately, Radio Shack does not list some of the parts needed for this modification, so unless your local store happens to carry parts that are not in the catalog, you will have to seek another parts supplier. The parts I used are show in Table 1.

1	Sylvania transistor	ECG 124
1	Sylvania socket	ECG 421
1	Sylvania zener diode	ECG 5050
1	18k 1W resistor	

Total cost should not exceed $5.

Table 1. *Parts list*

The regulator circuit is wired as shown in Figures 2 and 3. The 18k resistor serves as a current limiting resistor for the zener diode. The zener holds the base of the regulator transistor at 100 V dc. The transistor's emitter will always be within .6 volts dc of the base voltage. The 130-Ohm, 7-Watt resistor, which is located after the rectifier (Figure 2), distributes the supply voltage which is in excess of the 100 volt output of the regulator.

Short the 22-Ohm resistor (Figure 2**) with a piece of wire. Shorting this resistor allows the regulator to function over a greater range of line voltage variations.

Modification Tips

Consider the following precautions:

1. Be sure to unplug the power cord before you work on the monitor.

2. When installing the transistor, be sure to use the mica insulator and the two insulating washers supplied with the transistor. These isolate the transistor from the chassis. Be sure that the transistor is isolated.

3. Use a silicone-based heat-sink compound between the transistor and the mica, and between the mica and the chassis. The silicone ensures proper heat dissipation.

4. Use caution when working around the exposed CRT (picture tube). A sharp blow on the neck of the tube could cause an implosion, which would be, at the least, costly—not to mention dangerous. Place a large towel or heavy cloth over the tube while it is exposed; this will protect you in case of accident.

5. If you have a voltmeter, turn the power on and check the voltages (as indicated in Figure 2) before installing the "new" wire from the transistor. If

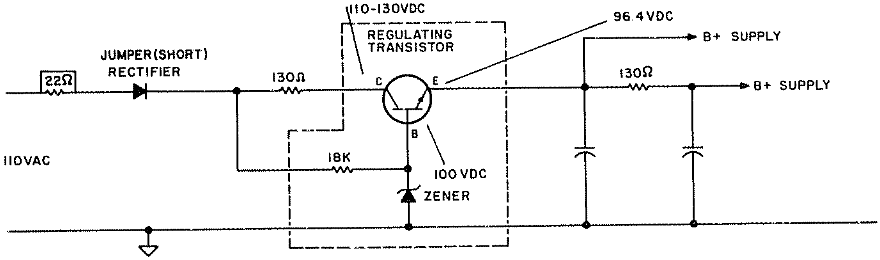


Figure 2. *Modified circuit*

the voltages don't match those in Figure 2, turn off the power and recheck all the connections. Make sure that the transistor is not shorting to the chassis.

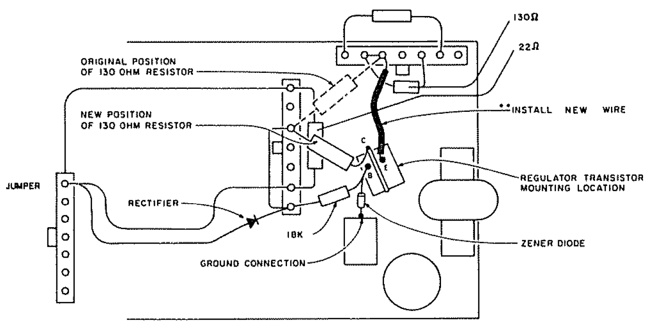


Figure 3. *Circuit modification*

CTR-80 Modifications

by John Simmons

Here's how to modify your CTR-80 to make it much more convenient to use with your TRS-80. The modifications are very simple and take less than an hour. First, make sure your recorder has been modified to prevent partial erasure of programs. This modification will be done free for the asking by Radio Shack; see your dealer. This modification is vital—don't overlook it and learn the hard way by losing important programs.

Before going on to the do-it-yourself modifications, be aware that they will void the 90-day Radio Shack warranty. They will also void the 30-day warranty on repairs or modifications done by Radio Shack. However, the first modification is so simple it can be removed in just a few minutes, should the need arise.

Modifications

The first modification allows you to hear what the computer is reading from the tape. All you'll need is a 33-Ohm, 1/2-Watt resistor, a small Phillips screwdriver, a pencil soldering iron, and some rosin core solder. Plug in the iron to warm it up and follow these steps:

1) Remove all cables and any cassette from the recorder. Place the recorder upside down with the jacks facing away from you on a flat surface. Put a soft cloth underneath the CTR-80 to protect it from scratches.
2) Remove the battery compartment door and set it aside. Remove the screw in the battery compartment and the two screws on the opposite end of the recorder.
3) Carefully separate the case by lifting up on the battery compartment half. There will be three wires leading from the battery compartment to the "guts" of the recorder; be careful not to break them if you are not going on to the next modification.
4) You will see that there are three solder pads on the PC board associated with the earphone (EAR) jack. Carefully solder the 33-Ohm resistor between the middle pad (the one that has a speaker wire soldered to it) and the pad farthest from the edge of the PC board. See Photo 1. Make sure that the resistor lies flat against the PC board and will not touch any other pads.
5) (Skip this step if you are going on to the next modification.) Reassemble in reverse order.

That's all there is to the first modification. Now, when you use the recorder with the computer, you will hear the program or data being read

by the computer. I find a 33-Ohm resistor produces just the volume I want. If you want more volume, try a 22-Ohm resistor; less volume, try 47 Ohms. You will have to use a slightly higher volume setting when loading a program—try one higher number. If you use the recorder for other purposes, the resistor has no effect when the earphone jack is not used.

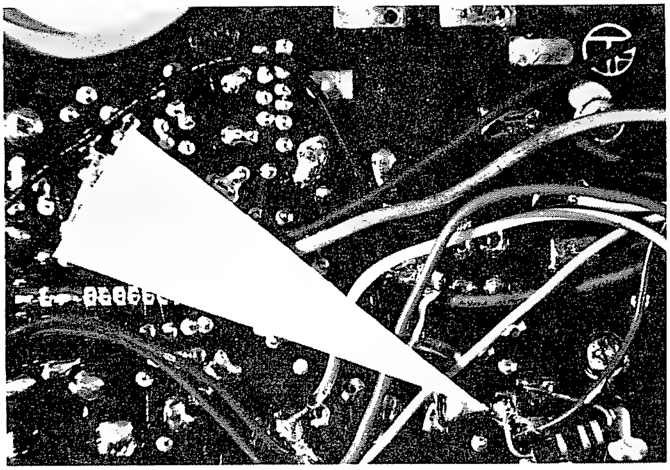


Photo 1. *33-Ohm resistor attached*

The next modification is only slightly more difficult and requires two things in addition to the tools needed above. You will have to sacrifice battery operation and the external six-volt (DC6V) jack. You will also need a sub-miniature SPST (single-pole single-throw) switch with two short lengths of wire. This modification allows manual play and record without unplugging the computer from the remote (REM) jack.

1) Follow steps 1-3 above.
2) Remove the three screws holding the recorder frame to the top half of the case. Remove the two screws holding the PC board. Be sure to note the hole each screw came from.
3) Remove the nuts from the microphone (MIC), remote, auxiliary, and earphone jacks.
4) Carefully pull the recorder slightly out of the case, enough to remove the two screws holding the jack panel to the recorder frame. Do not strain the wires leading to the condenser mike in the top half of the recorder case.
5) Remove the two screws holding the DC6V jack to the panel. Cut the three wires attached to the jack. Put the jack in your junk box.

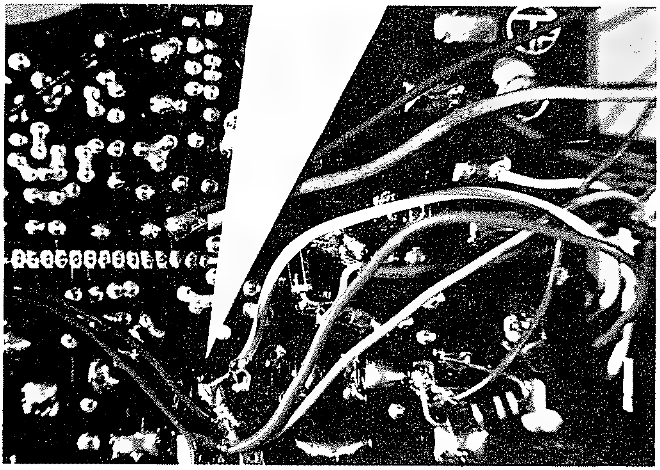


Photo 2. *Switch leads soldered to pads of the remote jack*

6) Solder a five-inch length of hookup wire to each switch terminal.
7) Install the switch in the vacated DC6V hole.
8) Solder the other ends of the switch leads to the two PC board pads of the remote jack. See Photo 2.
9) Reassemble in reverse order.

Now, with the switch in the on position, you may manually play a tape to locate the beginning of a program without removing any plugs. With the

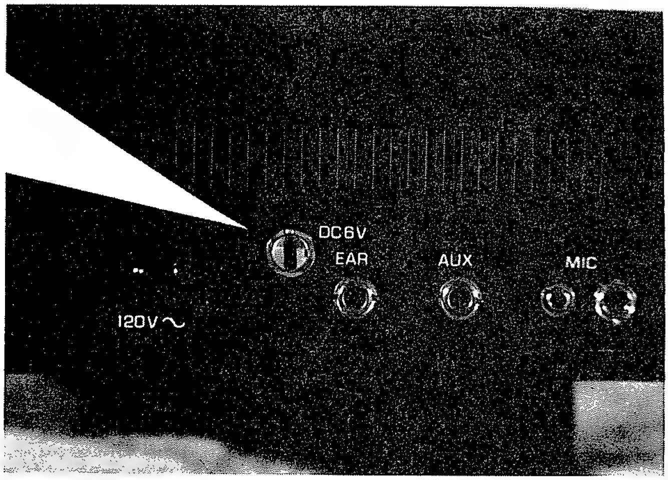


Photo 3. *Switch installed*

switch in the off position, the computer has total control of the recorder, as before. When I dump a program to tape, I rewind the cassette fully and put the CTR-80 in the record mode. Then I flip the newly installed switch to manual and let the tape run to 10 on the index counter. This insures that my new program will be the first on the tape, and bypasses any bad tape at the beginning of the cassette. I then turn the switch off, and start the dump (hit ENTER). When the computer is finished, I flip the switch on and advance the tape (still in the record mode) to the next round number. Then I flip the switch off and dump another (backup) copy of the program to tape. This adds a large safety margin.

HOME APPLICATIONS

The Great Girl Scout
Cookie Caper

Two Energy Savers

HOME APPLICATIONS

The Great Girl Scout Cookie Caper

by James N. Devlin

How many times have you gone to an innocent organizational meeting with your little boy or girl only to return home as the new cubmaster or district leader? Scout leaders are not born, they are made in meetings. My wife came home from just such a meeting and lo and behold, she had just been volunteered for the job of area cookie chairperson. Each of us had been similarly volunteered for many jobs in the past. I ended up as a cubmaster once and I even became a girl scout troop leader for several years as the result of just such organizational diplomacy.

At the time we didn't give it a lot of thought. Later however, when the cold light of reality dawned, the job began to take on ominous dimensions involving *math*, a subject that is not the most popular in the world, ranking somewhat above a poke in the eye in many people's minds.

What could be more natural in the face of such a crisis than to enlist the aid of the faithful TRS-80 lurking quietly in the corner of the room? It turned out not to be such an overwhelming task after all. There were only eight troops with a total of 120 girls along with a selection of seven types of cookies to be tallied by cases and boxes along with their various dollar amounts and totals. But the challenge was just too much to dismiss. A program had to be written!

This program (see Program Listing) can be adapted easily to a great number of similar situations where items are sold by members of an organization, and where there is a need to assemble the individual sales into a special format for purposes of ordering or to provide a tally of the results.

The program input consists of the boxes of different cookies sold by each of the girls. In our case there were seven varieties of cookies to be accounted for. These were placed into the program by means of a DATA READ statement, since next year there might be a different number of varieties or they might have different names. By placing the names of different items into this statement the program can be adapted readily to any group's particular requirements.

This was the approach that was taken for all parameters: the number of troops, the number and names of the girls in each of the troops, etc. These data statements are placed in the 900 and 1000 blocks so that the program can be easily tailored to individual troop situations. The girls' names were entered along with their respective troops and then read into a matrix. The size of the matrix is adjusted to fit using the troop size data in line 900 and 1000.

Line 900 contains the total number of troops, and the 1000 series of lines contains the total kids for that troop in each third element. By changing these numbers the program will compute any size group (within memory limits of course), distributed in any arbitrary way. However you must be consistent in the succeeding data to be sure that the number of names matches the number assigned to each troop. I did this by placing one troop in the 1100 block, for example, and another troop in 1200, etc.

When the girls turn in their cards, the information is simply transferred to the program when it is requested. For each troop, every girl's name is sequentially displayed, followed by each type of cookie. Then the number of boxes that she has sold of that particular type is entered. This procedure continues until all of the sales for all of the girls in that troop have been entered. The program then returns to the calling menu which presents the various options that are available. These options are for an input mode and a variety of output modes. The input mode permits the choice of either keyboard input or tape input. When keyboard input is requested, any of the troops listed can be selected individually and incorrect troop numbers are automatically rejected.

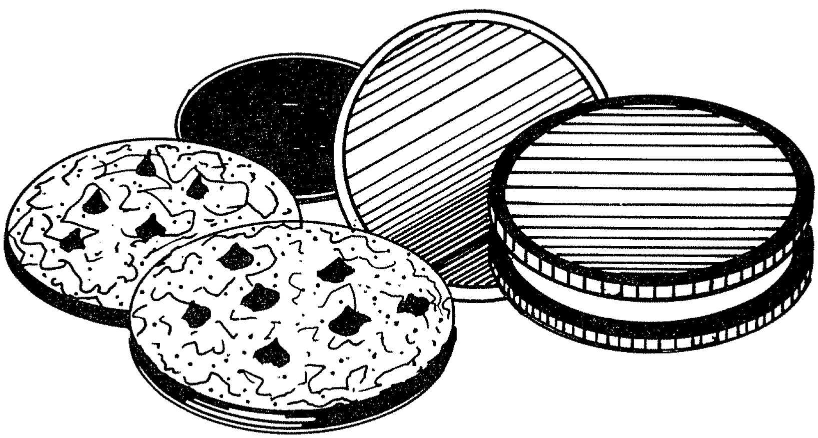
This is all of the information that is required, and it is the only record that needs to be kept by the individual troop leaders. All other information will appear on the display (or printout if you have a printer). All box-to-case conversions will be done by the program as well as the dollar amounts and troop profit.

As with any accounting or inventory program, some form of hard copy is desirable. Otherwise you must note down the essential output information from the screen. The program is presented with the normal print statements and it is a simple matter to replace the appropriate ones with LPRINT. If you are only concerned with case totals and summary information, hard copy is not needed.

After the information is entered for each of the girls, you can then select the troop results or the summary from the option billboard. Should you select the individual troop information, the screen will list the troop number and leader at the top. Then, in column form, it will present each girl's name followed by the number of boxes of each type of cookie that she sold. The total box count and the total money that she will have to collect from her customers follows in the right hand column. If more girls are in the troop than can be displayed on the screen, the table is held until the input key is depressed. Eight girls' names are displayed in the present program, but this can be changed in line 452.

After all of the girls and their totals have been displayed, a summation by cookie type will be presented, followed by the troop's total dollars and the troop's profit. This is an excellent document to return to the troop leader when the cookies are delivered, as it is a complete record of each

girl's sales performance. After printing the troop data, the machine computes the case totals and the extra boxes of each kind of cookie. This is the information that is needed when the troop leader comes to pick up her order and is really the bottom line of the troop data. It makes it very easy to sort out the proper individual deliveries from the grand total of wall-to-wall cases that will be filling your living room. If you don't have a printer, you can run the program at delivery time and display the individual troop records and case totals when the respective leaders come to pick up their cookies. Any troop's information can be selected at any time.



When selected, the summary sheet lists each troop number in the left hand column and each cookie type in the column headings. The cookie types are subdivided into the specific cases and boxes. In the right hand column is the total troop dollar figure. Here the individual troop performance can be compared and evaluated. At the bottom is the summation of actual total cases and boxes along with the total dollars that each troop is responsible for.

An output option lets you put all of the sales data onto a data tape so that you can make future runs of the program without having to manually input data again. As you are probably aware, the cookie supplier usually requires that all the individual troops place their orders in even case lots. This is where the program makes its greatest contribution. No longer will each of the troops have to convert its own sales into whole case lots. The program does this all at one time.

As an example of this advantage, take our own case where there were eight troops and seven separate varieties of cookies. If each troop (in the worst case) sold one extra box of each type of cookie, they would have to

order one extra case of that type of cookie just to cover the one box that was sold. This would result in each troop having to sell an additional eleven boxes of each kind of the seven varieties, or a total of 77 boxes more than they had already sold. If this were the situation for each of the eight troops, that comes to a total of 616 more boxes of cookies that must be sold in a territory that has already been saturated. In other words, 50 additional cases of cookies would have to be sold. That's great for the supplier, but not so good for the hard working little girls. If there were fifteen troops involved, that would end up being 96 cases or a phenomenal 1155 boxes.

Of course, not all troops will sell just one extra box of each type, but the residuals would be approximately half of this worst-case figure in the average situation. With the program and a single total computation the worst case figure dwindles to a mere 77 boxes or six cases and one could expect that this would average out to around three additional cases. Quite a difference!

With modifications, this program can be adapted to many other selling projects since the size of the matrix is read in as part of the data, along with the names of the items to be sold, each time that the program is run. Other products used to raise money by direct sales to consumers would simply be entered in the custom program. A commercial venture run from the home such as Amway would also lend itself to a program of this type.

In the Program Listing all of the troop members are listed by their initials, however in the program for our individual case the last names of each of the girls were used. This is a nice touch since kids love to see their names in print. Line 450 will print 13 characters of each name. The header READ statements in lines 100–180 automatically adjust the size of the matrix to accommodate the varying size of the troops. The eight troops of 120 scouts fit comfortably in the 16K machine and additional memory space can be obtained by shortening the names of the individuals to conserve string usage.

Next year, when you "volunteer" to take responsibility for your organization's favorite fundraiser, be ready and waiting with this program.

TROOP # 101 *** LEADER R.G.

NAME	CRS	CHE	CRE	TEA	CHI	MIN	SUG	TOTAL	AMNT
K.A.	4	1	2	10	6	16	5	44	66.00
M.B.	1	0	0	1	1	8	2	13	19.50
CI.F.	1	0	2	1	5	5	6	20	30.00
CH.F.	0	0	5	5	1	5	6	22	33.00
S.G.	5	1	3	9	6	14	9	47	70.50
K.J.	1	4	15	16	8	39	30	113	169.50
N.K.	12	4	6	20	21	24	19	106	159.00
C.Q.	3	4	4	12	9	25	12	69	103.50
R.R.	1	0	1	10	4	8	16	40	60.00
L.R.	3	1	8	12	8	17	16	65	97.50
A.S.	5	8	13	15	16	37	27	121	181.50
TOTALS	36	23	59	111	85	198	148	660	990.00

TROOP PROFIT IS 132.00

TROOP 1 CASE TOTALS

CRSP............	3 CASES	0 BOXES
CHES...........	1 CASES	11 BOXES
CREM..........	4 CASES	11 BOXES
TEAS...........	9 CASES	3 BOXES
CHIP............	7 CASES	1 BOXES
MINT...........	16 CASES	6 BOXES
SUGR...........	12 CASES	4 BOXES

*** COMPLETE SUMMARY ***

TROOP	CRSP	CHES	CREM	TEAS	CHIP	MINT	SUGR	AMNT
	C B	C B	C B	C B	C B	C B	C B	
101	3, 0	1,11	4,11	9, 3	7, 1	16, 6	12, 4	$ 990.00
373	4, 1	4, 8	6, 4	10,10	8, 6	29, 2	17, 7	$1461.00
512	1, 6	1, 7	3, 1	5, 7	3, 4	14, 2	7, 0	$ 652.50
533	2, 6	1, 7	3, 1	6,10	4,10	12,10	9, 3	$ 736.50
1210	5, 3	5,10	6, 1	14, 6	14, 3	34, 6	24, 9	$1893.00
1219	1,10	2,11	3, 8	5,10	5, 4	16,11	10, 4	$ 843.00
1235	1, 4	0, 8	1, 8	2, 6	1,10	6, 0	3,11	$ 322.50
1448	2, 7	1,11	2, 3	7,11	4, 2	14, 4	11, 6	$ 804.00
TOTALS	23	22	32	64	50	145	97	
RESID'S	11	11	11	9	8	7	4	

TOTAL CASE COUNT = 433 CASES.

TOTAL DOLLAR VALUE IS $7794.00

Figure 1. *Sample run*

Program Listing

```
 5 DIM MM$(12Ø),A1(12Ø),B1(12Ø),C1(12Ø),D1(12Ø),E1(12Ø),F1(12Ø),G1(
   12Ø),S2(12Ø)
 7 FOR I = 1 TO 3:
    OUT 1,Ø:
    NEXT I
 8 OUT 1,64:
   OUT 1,25Ø:
   OUT 1,51
 1Ø REM  G.S. COOKIE PGM -J.DEVLIN ,79
 25 CLS :
    PRINT :
    PRINT "** GIRL SCOUT COOKIE PROGRAM **"
 3Ø PRINT :
    PRINT :
    PRINT :
    PRINT "WANT INSTRUCTIONS ?"
 32 INPUT "YES/NO";Y$
 35 IF Y$ = "YES"
     THEN
      5ØØ
 4Ø INPUT "CHANGE PRINT MODE (YES/NO)";Y$
 45 IF Y$ = "YES"
     THEN
      GOSUB 32ØØØ
 5Ø READ T
 6Ø SS = Ø:
    A$ = "####.##":
    B$ = "##":
    B1$ = "###"
1ØØ FOR I = 1 TO 7:
     READ C$(I):
     NEXT I
15Ø FOR I = 1 TO T:
     READ TN(I):
     READ TL$(I):
     READ TT(I):
     SS = SS + TT(I)
16Ø  TE(I) = SS:
     TS(I) = TE(I) - TT(I) + 1:
     NEXT I
17Ø FOR I = 1 TO T
18Ø  FOR J = 1 TO SS:
      READ MM$(J):
      NEXT J
2ØØ  CLS :
     PRINT TAB(1Ø),"DATA INPUT..........(1)"
2Ø1  PRINT TAB(1Ø),"TROOP SUMMARY.......(2)"
2Ø3  PRINT TAB(1Ø),"COMPLETE SUMMARY....(3)"
2Ø4  PRINT TAB(1Ø),"END.................(4)"
2Ø5  PRINT :
     PRINT "VALID TROOP NUMBERS":
     FOR I = 1 TO T:
      PRINT "TRP-";I,TN(I):
      NEXT I
2Ø8  INPUT "ENTER # OF OPERATION";R
21Ø  IF R > Ø AND R < 5
      THEN
       23Ø
22Ø  PRINT "INVALID ENTRY":
     GOTO 2Ø8
23Ø  ON R GOTO 24Ø,4ØØ,8ØØ,7ØØ
24Ø  INPUT "WILL DATA ENTERED FROM TAPE (1) OR KEYBOARD (2)";Y
25Ø  IF Y = 2
      THEN
       3ØØ
26Ø  INPUT "TAPE LOADED & PLAY SET... HIT ENTER";X
265  CLS :
```

```
     PRINT CHR$(23);:
     PRINT @266,"LOADING"
270  FOR I = 1 TO SS
275   INPUT # - 1,A1(I),B1(I),C1(I),D1(I),E1(I),F1(I),G1(I):
      PRINT @468,I
280   NEXT I
285  FOR K = 1 TO T
290   GOSUB 360
295   NEXT K:
     GOTO 200
300  INPUT "WHICH TROOP";N
302  FOR I = 1 TO T:
      IF N = TN(I)
       THEN
        K = I:
        GOTO 310
304   NEXT I
305  PRINT "TROOP NOT IN MY MEMORY-TRY AGAIN":
     GOTO 300
310  CLS :
     PRINT :
     PRINT "ENTER THE # OF BOXES SOLD BY EACH GIRL"
314  PRINT :
     INPUT "IF NO CHANGE -HIT ENTER";X
315  FOR L = 1 TO 6:
      S1(L,K) = 0:
      NEXT L
316  CLS :
     FOR I = TS(K) TO TE(K):
      PRINT MM$(I)
318   INPUT "ENTER NONE TO SKIP ,ELSE ENTER";Y$:
      IF Y$ = "NONE"
       THEN
        338
320   FOR J = 1 TO 7:
       PRINT C$(J):
       INPUT A
330    ON J GOSUB 340,341,342,343,344,345,346
335    NEXT J:
      INPUT "-HIT ENTER";X
338   Y$ = "Y":
      CLS :
      NEXT I:
     GOTO 350
340  A1(I) = A1(I) + A:
     RETURN
341  B1(I) = B1(I) + A:
     RETURN
342  C1(I) = C1(I) + A:
     RETURN
343  D1(I) = D1(I) + A:
     RETURN
344  E1(I) = E1(I) + A:
     RETURN
345  F1(I) = F1(I) + A:
     RETURN
346  G1(I) = G1(I) + A:
     RETURN
348  FOR I = 1 TO 7:
      S(I,K) = 0:
      NEXT I
350  GOSUB 360
355  GOTO 200
360  FOR J = TS(K) TO TE(K)
365   S1(1,K) = S1(1,K) + A1(J):
      S1(2,K) = S1(2,K) + B1(J):
      S1(3,K) = S1(3,K) + C1(J)
370   S1(4,K) = S1(4,K) + D1(J):
      S1(5,K) = S1(5,K) + E1(J):
      S1(6,K) = S1(6,K) + F1(J):
```

Program continued

```
      S1(7,K) = S1(7,K) + G1(J)
375   S2(J) = A1(J) + B1(J) + C1(J) + D1(J) + E1(J) + F1(J)
      + G1(J):
      NEXT J
378  GT(K) = 0:
     FOR I = 1 TO 7:
      GT(K) = GT(K) + S1(I,K)
380   S3(I,K) = S1(I,K) / 12:
      S4(I,K) = INT(S3(I,K)):
      U = (S3(I,K) - S4(I,K)) * 12:
      S5(I,K) = INT(U + .5)
385   NEXT I
390  RETURN
400  INPUT "WHICH TROOP ";N
404  Z = 1:
     FOR I = 1 TO T:
      IF N = TN(I)
       THEN
        K = I:
        GOTO 410
405   NEXT I
406  PRINT "THAT TROOP NOT IN MY MEMORY-TRY AGAIN":
     GOTO 400
410  CLS :
     PRINT "TROOP #";TN(K);:
     PRINT " ***   LEADER "TL$(K):
     PRINT
420  PRINT "NAME              ";:
     FOR I = 1 TO 7:
      PRINT LEFT$(C$(I),3);"  ";:
      NEXT I:
     PRINT " TOTAL    AMNT"
430  PRINT
440  FOR J = TS(K) TO TE(K)
450   PRINT LEFT$(MM$(J),13);:
      PRINT TAB(14)A1(J); TAB(19)B1(J); TAB(24)C1(J); TAB(29)D1(J);
      TAB(34)E1(J); TAB(39)F1(J); TAB(44)G1(J); TAB(50)S2(J);
      TAB(55):
      PRINT USING A$;S2(J) * 1.50
452   IF Z > 9
       THEN
        INPUT "HIT INPUT TO CONTINUE";X:
        Z = 1:
        CLS :
        GOTO 455
453   Z = Z + 1
455   NEXT J:
     PRINT
460  PRINT TAB(0)"TOTALS"; TAB(14)S1(1,K); TAB(19)S1(2,K); TAB(24)S1
     (3,K); TAB(29)S1(4,K); TAB(34)S1(5,K); TAB(39)S1(6,K);
     TAB(44)S1(7,K); TAB(50)GT(K); TAB(55):
     PRINT USING A$;GT(K) * 1.50
465  PRINT :
     PRINT "TROOP PROFIT IS ";:
     PRINT USING A$;GT(K) * .20
470  INPUT "HIT INPUT FOR CASE TOTALS";X:
     CLS :
     PRINT
472  PRINT :
     PRINT "TROOP ";K;" CASE TOTALS"
475  PRINT :
     FOR I = 1 TO 7
480   PRINT TAB(0)C$(I);".......";  TAB(20)S4(I,K);"CASES"; TAB(30)S5
      (I,K);"BOXES"
485   NEXT I
495  INPUT "-HIT ENTER";X:
     GOTO 200
500  CLS :
     PRINT :
     PRINT "THIS PGM OPERATES FROM LISTS OF TROOPS."
```

```
540  PRINT "1000 DATA TRP # 1,LDR 1,15, ETC."
590  PRINT "LINES 1100-1999 CONTAIN GIRL'S NAMES"
650  PRINT "LINE 900 = TOTAL # OF TROOPS"
660  PRINT "LINE 910 = COOKIE TYPES"
695  PRINT :
     INPUT "-HIT ENTER";X:
     GOTO 50
700  INPUT "DO YOU WISH TO RECORD THIS DATA YES/NO";Y$
710  IF Y$ = "YES"
      THEN
       740
720  GOTO 785
740  INPUT "IF TAPE LOADED, CUED AND IN RECORD... HIT ENTER";X
750  FOR I = 1 TO SS
760   PRINT # - 1,A1(I),B1(I),C1(I),D1(I),E1(I),F1(I),G1(I)
770   NEXT I
780  PRINT :
     PRINT "*TAPE COMPLETE*"
785  INPUT "CHANGE PRINT MODE (YES/NO)";Y$
787  IF Y$ = "YES"
      THEN
       32000
790  INPUT "ANOTHER LOOK AT THE DATA....YES/NO";Y$
795  IF Y$ = "YES"
      THEN
       200
798  GOTO 9999
800  REM  SUMMARY PRINT OUT
810  CLS :
     PRINT :
     PRINT "               ***  COMPLETE SUMMARY  *** "
820  PRINT :
     PRINT "TROOP  ";:
     FOR I = 1 TO 7:
      PRINT " "; LEFT$(C$(I),4);"  ";:
      NEXT I:
     PRINT "   AMNT"
825  FOR I = 1 TO 7:
      PRINT TAB(I * 7)" C  B";:
      NEXT I
827  PRINT
830  FOR K = 1 TO T
840   PRINT TN(K);:
      FOR J = 1 TO 7:
       PRINT TAB(J * 7):
       PRINT USING B$;S4(J,K);:
       PRINT ",";:
       PRINT USING B$;S5(J,K);:
       NEXT J:
      PRINT " $";:
      PRINT USING A$;GT(K) * 1.50
850   NEXT K
852  FOR J = 1 TO 7:
      T1(J) = 0:
      NEXT J
855  FOR J = 1 TO 7:
      FOR K = 1 TO T:
       T1(J) = T1(J) + S1(J,K):
       NEXT K:
      NEXT J
860  FOR J = 1 TO 7:
      T7(J) = T1(J) / 12:
      T8(J) = INT(T7(J)):
      T6(J) = (T7(J) - T8(J)) * 12:
      T9(J) = INT(T6(J) + .5):
      NEXT J
865  FOR J = 1 TO 7:
      IF T9(J) < = 0
       THEN
        868
```

Program continued

```
867   T8(J) = T8(J) + 1:
      T9(J) = 12 - T9(J)
868   NEXT J
870  PRINT :
     PRINT "TOTALS";:
     FOR J = 1 TO 7:
      PRINT TAB(J + 6):
      PRINT USING B1$;T8(J);:
      PRINT "     ";:
      NEXT J
875  PRINT :
     PRINT "RESID'S";:
     FOR J = 1 TO 7:
      PRINT TAB(J + 7):
      PRINT USING B$;T9(J);:
      PRINT "     ";:
      NEXT J
880  S6 = 0:
     FOR J = 1 TO 7:
      S6 = S6 + T8(J):
      NEXT J
885  PRINT :
     PRINT :
     PRINT "TOTAL CASE COUNT= ";S6;" CASES.
890  PRINT :
     PRINT "TOTAL DOLLAR VALUE IS  $ ";:
     PRINT USING A$;S6 * 18.0
895  INPUT "-HIT ENTER";X:
     GOTO 200
900  DATA 8
910  DATA CRSP,CHES,CREM,TEAS,CHIP,MINT,SUGR
1000 DATA 101,R.G.,11,373,J.S.,20,512,N.T.,16
1010 DATA 533,D.H.,16
1020 DATA 1210,P.S.,23,1219,E.D.,19
1030 DATA 1235,L.W.,3,1448,N.R.,12
1100 DATA K.A.,M.B.,CI.F.,CH.F.,S.G.
1110 DATA K.J.,N.K.,C.Q.,R.R.,L.R.
1120 DATA A.S.
1200 DATA K.A.,E.A.,N.B.,J.D.,M.D.
1210 DATA L.G.,M.G.,J.K.,K.K.,C.L.
1220 DATA S.M.,J.MCG.,L.P.,B.P.
1230 DATA M.P.,K.R.,L.V.,M.W.,M.W.,T.W.
1300 DATA G.B.,M.B.,A.B.,T.E.
1310 DATA D.F.,C.G.,D.MCK.,T.O'B.
1320 DATA K.P.,S.P.,J.R.,A.S.
1330 DATA J.W.,T.W.,A.W.,S.W.
1400 DATA K.B.,C.C.,W.F.,S.G.
1410 DATA K.G.,L.H.,L.K.,J.M.
1420 DATA M.N.,D.P.,K.R.,M.R.
1430 DATA T.R.,M.S.,K.W.,S.Z.
1500 DATA C.B.,M.B.,K.C.,J.D.
1510 DATA C.D.,L.F.,M.F.,A.H.
1520 DATA M.H.,C.J.,R.J.,B.K.
1530 DATA T.K.,M.K.,M.M.
1540 DATA L.P.,K.R.,K.S.,H.S.
1550 DATA J.S.,B.S.,R.U.,S.W.
1600 DATA B.A.,T.C.,B.D.,M.E.
1610 DATA S.E.,J.G.,L.MCC.,S.MCG.
1620 DATA R.R.,H.S.,T.S.,K.V.
1630 DATA K.W.,E.W.,R.W.,A.W.
1640 DATA M.B.,T.N.,S.V.
1700 DATA S.G.,A.M.,L.R.
1800 DATA M.B.,C.C.,T.D.,K.D.
1810 DATA D.F.E,C.G.,M.M.,C.M.
1820 DATA S.S.,L.S.,D.V.,T.W.
9999  END
32000 INPUT "ENTER LPRINT FOR TTY";A$
32010 IF A$ = "LPRINT"
       THEN
```

```
        A = 175:
        B = 178
32020  IF A$ < > "LPRINT"
        THEN
        A = 178:
        B = 175
32030  FOR M = 17129 TO 28671
32040   IF PEEK(M) = B AND PEEK(M + 1) = 32
        THEN
         POKE M,A
32050   NEXT M:
       RETURN
```

Two Energy Savers

by Joseph H. Hart

Saving on energy bills (electricity, oil, gas, etc.) is always a popular topic of conversation. If you are like me, you have tried different ways to save money on your energy bills, but you may not know how much you have actually saved. In this chapter, I will discuss two programs that can help you figure out your energy savings.

ENERSAVE (Program Listing 1) will calculate your energy savings from the information you enter. I have found that more than just instructions are an absolute necessity in programs of this type. Therefore, I have included an explanation and definitions, as well as instructions within the program.

After asking if you want instructions, ENERSAVE asks you about your home. What type of heating do you have? Do you have an air conditioner? What is the present R-value of the insulation in the area you are going to insulate? (Some terms may be new to you, so I have included a list of definitions in Table 1). Next, ENERSAVE asks about cost. How much does your heating fuel cost? How much does your cooling fuel (electricity) cost? Then it asks you about the area to be insulated. Where are you going to insulate—ceiling, wall, or floor? What is the square footage of the area to be insulated? What is the R-value of the insulation you are going to add?

DEFINITIONS

Heating Degree-Days—an indication of the need for heating. Obtainable from the local National Weather Bureau.
Cooling Hours—an indication of the time cooling is needed.
R-Value—a measure of the ability of a substance to resist the flow of heat.

Table 1. *Definitions*

To be able to accurately calculate the savings over the life of the insulation, I had to find a way to account for the annual increase in fuel cost, as well as the amount of interest if you took out a loan to buy and install this insulation. ENERSAVE will ask you for these percentages. It will then calculate your savings as an equal annual savings for the life of the insula-

tion. I have found it easier to have this information collected ahead of time, so to help you, I have included a list of the necessary information in Table 2.

NECESSARY INPUT

DD	Heating degree-days
CH	Number of cooling hours
EH	Heat energy cost
EC	Cooling energy cost
II	R-value of insulation to be added
PI	R-value of existing insulation
N	Expected life of new insulation
I	Cost of money to you (interest rate)
E	Expected annual increase in energy cost (percentage)
SF	Square footage of area to be insulated

OPTIONAL INPUT

IC	Cost of insulation
IS	Cost to install

Table 2. *Information needed to use ENERSAVE*

Now that you know how much you will save annually, you may want to know how many years it will take to recover your investment. ENERSAVE will ask you for the cost of the insulation and the cost of installing this insulation. It will then calculate the number of years to recover your investment.

How Much Did You Save on Your Electric Bill Last Year?

ELECTCOM (see Program Listing 2) will tell you the amount of electrical energy you used for heating, cooling, base, and the savings for each year. This sounds like it would be very simple to do, but it really isn't. Let's say you want to find how much you saved on electricity by trying to compare the total dollars spent for electricity month by month for each year. This method is not accurate, because you have compared the cost of electricity for one year to a higher cost of electricity for the next. For example, electricity increased about 34 percent in 1979. You have not compared your electricity *use*. Another reason why it is not accurate to compare energy on a cost basis alone is the fact that one year may be colder or hotter than the next. If you compare a cold winter with a mild winter, there is a savings for the milder winter, just because less heat energy was required. The same ef-

fect occurs when you compare a hot summer to a mild summer. Less cooling energy is needed in the milder summer than in the hotter summer.

How do we compare one year's electrical use to another? We compare energy use, or kilowatt-hours. Now we are comparing the energy used in one year compared to another, independent of cost. After we determine the savings in energy, we can multiply by the average cost of electrical energy to obtain cost savings.

We still need to make an adjustment for the fact that winter may be colder one year than the other and summer may be hotter one year than the other. To achieve this, I have used something called a degree-day in this program. A degree-day is a unit that represents the amount of heating or cooling energy needed. Heating and cooling degree-days for your area can be obtained from your local National Weather Bureau. The degree-days are calculated daily and used to determine the monthly and yearly degree-days for both heating and cooling. Therefore, the heating degree-days and cooling degree-days change from year to year.

Now that we know how the program adjusts for heating and cooling and variations due to weather, and that the program uses kilowatt-hours for comparison, all we need is electricity usage for each month. You will probably find that your monthly electrical bills are not always (if ever) on a calendar month basis, that is, usage from the first of the month to the end of the month, but that doesn't matter. Just use the kilowatt-hour usage on each monthly bill. It doesn't matter what month with which you start, as long as the annual heating and annual cooling degree-days match. Therefore, you may use any twelve month period for the first year and any other twelve month period for the second year, as long as the annual heating and cooling degree-days correspond for the two twelve month periods.

I hope these programs will help you to use energy in the most efficient manner.

Program Listing 1. *ENERSAVE*

```
 1Ø REM   * INSULATION , PAYBACK AND SAVINGS EVALUATION *        *
    VERSION 1.1   - - -  MARCH 2, 198Ø  - - -    *
 2Ø REM   *    COPYRIGHT (C)  198Ø    *         *       JOSEPH H. HA
    RT       *            *  2312 THOUSAND OAKS DR.    *        *    RI
    CHMOND  VIRGINIA     *
 3Ø CLS :
    PRINT :
    PRINT :
    PRINT :
    PRINT CHR$(23); TAB(4) STRING$(25,"* ");"    *       INSULATION
         *     * PAYBACK AND SAVINGS   *    *      EVALUATION         *
    "
 4Ø PRINT TAB(4) STRING$(25,"*"):
    FOR X = 1 TO 1ØØØ:
     NEXT
 5Ø PRINT :
    PRINT :
    INPUT "DO YOU WISH INSTRUCTIONS";G$:
    IF G$ = "NO" OR G$ = "N"
     THEN
      1ØØØ
 59 REM   *  INSTRUCTIONS  *
 6Ø CLS :
    PRINT "       THIS PROGRAM WILL ESTIMATE YOUR ANNUAL MONEY SAVING
    SRESULTING FROM THE INSTALLATION OF INSULATION.  IT USES BASICEN
    ERGY EQUATIONS FROM 'A S H R A E' (AMERICAN SOCIETY OF          "
 7Ø PRINT "HEATING, REFRIGERATING AND AIR CONDITIONING ENGINEERS) TO
    CALCULATE THE HEAT LOSS AND/OR HEAT GAIN.  IT ALSO USES ECONOMIC
    TERMS ASSOCIATED WITH MOVING THE VALUE OF MONEY FROM THE FUTURE
    BACKWARDS TO THE PRESENT.  THESE TERMS ARE ";
 8Ø PRINT "CALLED 'CAPITALRECOVERY FACTOR' AND 'PRESENT WORTH FACTOR
    '.  I HAVE TAKEN THE  'PRESENT WORTH FACTOR' AND CONVERTED IT IN
    TO A 'GEOMETRIC SERIES' WHICH INCORPERATES AN ESCALATION  FACTOR
    .  THERFORE THE 'ESTIMATED ANNUAL HEATING AND";
 9Ø PRINT " COOLING SAVINGS FOR 'N' YEARS' ISA TYPE OF AVERAGE SAVIN
    GS FOR THE 'N' YEARS.":
    PRINT @9ØØ,"(HIT ENTER TO CONTINUE)";:
    INPUT G$:
    CLS
1ØØ PRINT "      BEFORE YOU CAN UTILIZE THIS PROGRAM, CERTAININFORMAT
    ION (NOT USUALLY KNOWN) WILL NEED TO BE GATHERED. THE   FOLLOWIN
    G NUMBERS WILL BE NEEDED:"
11Ø PRINT TAB(3)"ANNUAL DEGREE DAYS (IF KNOWN)"; TAB(4Ø)"TYPE OF HEA
    T"
14Ø PRINT TAB(3)"R-VALUE OF PRESENT INSULATION"; TAB(4Ø)"VALUE OF MO
    NEY (%)"
15Ø PRINT TAB(3)"R-VALUE OF ADDITIONAL INSULATION"; TAB(4Ø)"ENERGY E
    SCALATION (%)"
16Ø PRINT "SQUARE FOOTAGE OF AREA TO BE INSULATED":
    PRINT @9ØØ,"(HIT ENTER TO CONTINUE)";:
    INPUT G$:
    CLS
17Ø PRINT :
    PRINT "DEFINITIONS:"
18Ø PRINT TAB(5)"ANNUAL DEGREE DAYS - A UNIT MEASURING THE EXTENTTO
    WHICH THE OUTDOOR MEAN (AVERAGE OF MAXIMUM AND MINIMUM) DAILYDRY
    -BULB TEMPERATURE FALLS BELOW 65 DEGREES F. TOTALED FOR EACH DAY
```

Program continued

```
120 PRINT TAB(3)"ANNUAL COOLING HOURS (IF KNOW)"; TAB(40)"A/C  (YES
    OR NO)
130 PRINT TAB(3)"HEATING COST ($#.###)"; TAB(40)"COOLING COST ($#.0#
    #)
    FOR ONE YEAR."
190 PRINT TAB(5)"ANNUAL COOLING HOURS - A UNIT MEASURING THE TIMETHE
     COMPRESSOR OF AN AIR CONDITONING SYSTEM RUNS PER YEAR."
200 PRINT TAB(5)"R-VALUE - A UNIT MEASURING THE RESISTENCE TO THE FL
    OW OF HEAT THROUGH A CERTAIN MEDIUM."
210 PRINT TAB(5)"ENERGY ESCALATION - THE RATE (%) AT WHICH ENERGY WI
    LL INCREASE IN COST PER YEAR."
220 PRINT :
    PRINT :
    INPUT "NOW YOU'RE READY TO BEGIN (HIT ENTER)";G$
999 REM   *  INPUT DATA ROUTINE   *
1000 CLEAR 2000:
    DEFSNG F,C,E,I,P,N,T:
    DEFINT A,B,S,D,X:
    CLS :
    PRINT :
    INPUT "IF YOU KNOW THE NUMBER OF DEGREE DAYS FOR YOUR AREA, ENTE
    R IT;BUT IF YOU DON'T ENTER THE NUMBER '4000'";DD:
    GOSUB 4000:
    IF Z = 1 GOTO 1000
1010 PRINT "WHAT TYPE OF ENERGY DO YOU USE TO HEAT YOUR HOME?
            HEAT PUMP - - - - - - - - - 1                ELECTRIC BASE
    BOARD - - -  2                    ELECTRIC FURNACE - - - -  3
             GAS - - - - - - - - - - - - 4
1020 PRINT CHR$(27)"                    OIL - - - - - - - - - - - - 5":
    INPUT A
1029 REM   *  CALCULATING FUEL CONSTANT   *
1030 IF A = 1
     THEN
      FC = 60 * DD * 20 * .9 / 3413 / 60 / 1.8:
      TH$ = "HEAT PUMP"
1040 IF A = 2
     THEN
      FC = 60 * DD * 17 * .9 / 3413 / 60 / 1:
      TH$ = "ELECTRIC BASEBBOARD"
1050 IF A = 3
     THEN
      FC = 60 * DD * 20 * .9 / 3413 / 60 / 1:
      TH$ = "ELECTRIC FURNACE"
1060 IF A = 4
     THEN
      FC = 60 * DD * 24 * .9 / 60 / 100000 / .605:
      TH$ = "GAS"
1070 IF A = 5
     THEN
      FC = 60 * DD * 24 * .9 / 60 / 140000 / .54:
      TH$ = "OIL"
1080 IF A < 1 OR A > 5 GOTO 1010
1090 INPUT "DO YOU HAVE AN AIR CONDITIONER";AC$:
    IF AC$ < > "YES" IF AC$ < > "NO" PRINT "ANSWER YES OR NO !!!":
    GOTO 1090
1095 IF AC$ = "NO" GOTO 1110
1100 INPUT "IF YOU KNOW THE NUMBER OF COOLING HOURS FOR YOUR AREA,ENT
    ER IT; BUT IF YOU DON'T, ENTER THE NUMBER '650'";CH:
    GOSUB 4010:
    IF Z = 2 GOTO 1100
1110 CLS :
    PRINT "WHERE DO YOU PLAN TO ADD INSULATION?              CEILING -
     - - - - - 1              WALL - - - - - - - -  2           FLOOR -
     - - - - - - - 3":
     INPUT B
1120 IF AC$ = "NO"
      THEN
       1160
1129 REM   *  CALCULATING COOLING CONSTANT   *
1130 IF B = 1
```

```
      THEN
       CC = CH * 45 * .5 * .9 / 3413:
       P$ = "CEILING":
       GOTO 1160
1140 IF B = 2
      THEN
       CC = CH * 22 * .5 * .9 / 3413:
       P$ = "WALL":
       GOTO 1160
1150 IF B < > 3
      THEN
       1110:
      ELSE
       CC = 0 AND FC = FC / 2:
       P$ = "FLOOR"
1160 INPUT "HOW MUCH DOES YOUR HEATING ENERGY COST($0.0##/KWH OR$0.##
     #/CCF OR $(#.##/GAL.)";EH:
     GOSUB 4020:
     IF Z = 3 GOTO 1160:
     IF AC$ = "NO" GOTO 1180
1170 INPUT "HOW MUCH DOES YOUR COOLING ENERGY COST($0.0##/KWH)";EC:
     GOSUB 4030:
     IF Z = 4 GOTO 1170
1180 IF B = 1 GOSUB 3000
1190 IF B = 2 GOSUB 3100
1200 IF B = 3 GOSUB 3200
1210 INPUT "WHAT IS THE AVERAGE R-VALUE OF THE INSULATION YOU PLAN TO
            INSTALL";II:
     GOSUB 4040:
     IF Z = 5 GOTO 1210
1220 INPUT "WHAT IS THE APPROXIMATE AVERAGE R-VALUE OF THE INSULATION
      YOU PRESENTLY HAVE";PI:
     GOSUB 4040:
     IF Z = 11 GOTO 1220
1230 CLS :
     INPUT "HOW MANY YEARS DO YOU EXPECT THE INSULATION TO LAST";N:
     GOSUB 4080:
     IF Z = 9 GOTO 1230
1240 INPUT "IF YOU WERE TO BORROW THE MONEY FOR THE INSULATION AND TH
     EINSTALLAION,  WHAT WOULD BE THE INTEREST RATE (%)";I:
     GOSUB 4050:
     IF Z = 6 GOTO 1240
1250 INPUT "HOW MUCH DO YOU EXPECT ENERGY TO INCREASE EACH YEAR  (%)"
     ;E:
     GOSUB 4060:
     IF Z = 7 GOTO 1250
1260 INPUT "WHAT IS THE SQUARE FOOTAGE OF THE AREA TO WHICH YOU ARE G
     OING TO ADD INSULATION";SF:
     GOSUB 4070:
     IF Z = 8 GOTO 1260
1269 REM   *  CALCULATING 'CAPITAL RECOVERY FACTOR ' AND    *
           *  'GEOMETRIC SERIES PRESENT WORTH FACTOR '      *
1270 IX = I / 100:
     EX = E / 100:
     CRF = IX / (1 - (1 + IX) [ - N):
     GSPWF = 0:
     FOR K = 1 TO N:
      GSPWF = GSPWF + ((1 + EX) / (1 + IX)) [ K:
      NEXT
1279 REM   *  CALCULATING SAVINGS *
1280 EA = CRF * GSPWF * SF * ((1 / PI) - 1 / (PI + II)) * ((FC
     * EH) + (CC * EC))
2269 REM   * DISPLAYING INPUT AND SAVINGS ROUTINE  *
2270 CLS :
     X$ = "$$#.###":
     V$ = "$$###,###.##":
     Y$ = "###.#":
     Z$ = "####":
     T$ = "##.#":
     U$ = "###,###":
```

Program continued

```
     PRINT TAB(21)"INSULATION ANALYSIS":
     PRINT TAB(28)"SAVINGS":
2280 PRINT "USING THE FOLLOWING INPUT - - -":
2290 PRINT TAB(5)"DEGREE DAYS"; TAB(22)"=";:
     PRINT TAB(24) USING Z$;DD;:
     PRINT TAB(37)"COOLING HOURS"; TAB(55)"=";:
     PRINT TAB(57) USING Z$;CH
2300 PRINT TAB(5)"TYPE OF HEAT"; TAB(22)"="; TAB(24)TH$
2310 PRINT TAB(5)"AIR CONDITIONER"; TAB(22)"="; TAB(24)AC$;
2320 PRINT TAB(37)"COST OF COOLING"; TAB(55)"="; TAB(57) USING X$;EC;
2330 PRINT TAB(5)"COST OF HEATING"; TAB(22)"="; TAB(24) USING X$;EH;
2340 PRINT TAB(37)"INSULATION LIFE"; TAB(55)"="; TAB(57) USING T$;N
2350 PRINT TAB(5)"COST OF MONEY"; TAB(22)"="; TAB(24) USING T$;I;:
     PRINT "%"; TAB(37)"ENERGY ESCALATION"; TAB(55)"="; TAB(57)
     USING T$;E;:
     PRINT "%"
2360 PRINT "SQUARE FOOTAGE OF ";P$;" TO BE INSULATED IS "; USING U$;S
     F;:
     PRINT " SQ.FT."
2370 PRINT "PRESENT INSULATION IN THE ";P$;" HAS AN R-VALUE OF ";
     USING T$;PI;:
     PRINT "."
2380 PRINT "PLANNING TO ADD INSULATION WITH AN R-VALUE OF ";
     USING T$;II;:
     PRINT "."
2390 FI = PI + II:
     PRINT "FINAL R-VALUE FOR ";P$;" IS "; USING T$;FI;:
     PRINT "."
2400 PRINT TAB(10)"YOUR ESTIMATED ANNUAL HEATING AND COOLING SAVINGS
     FOR "; USING T$;N;:
     PRINT " YEARS IS ";:
     PRINT USING V$;EA;:
     PRINT ".":
     IF G = 5
      THEN
       2440
2410 PRINT TAB(20)"DO YOU WISH TO KNOW THE PAYBACK ";:
     INPUT M$:
     IF M$ = "NO"
      THEN
       PRINT @936,"EVALUATION TERMINATED.":
       END
2419 REM   *  PAYBACK ROUTINE  *
2420 CLS :
     G = 5:
     INPUT "HOW MUCH DO YOU THINK THE INSULATION WILL COST";IC
2430 INPUT "HOW MUCH DO YOU THINK IT WILL COST TO INSTALL THIS INSULA
     TION";IS:
     PB = (IC + IS) / EA:
     GOSUB 4090:
     GOTO 2270
2440 PRINT @896,"* * YOU WILL RECOVER YOUR INVESTMENT IN ABOUT ";:
     PRINT USING T$;PB;:
     PRINT " YEARS. * *":
     INPUT "(HIT ENTER FOR ANOTHER ANALYSIS)";M$:
     RUN 1000
2999 REM   * ROUTINE TO DISPLAY INSULATION R-VALUES  *
3000 CLS :
     PRINT TAB(28)"CEILING"
3010 PRINT TAB(18)"AVERAGE STRUCTUAL R-VALUE":
     PRINT TAB(22)"BASED ON INSULATION"
3020 PRINT "INSULATION R-VALUE"; TAB(47)"AVERAGE R-VALUE"
3030 FOR X = 1 TO 3:
      READ IR,AR:
      PRINT TAB(11)IR; TAB(53)AR:
      NEXT :
     FOR X = 1 TO 5:
      READ IR,AR:
      PRINT TAB(10)IR; TAB(52)AR:
      NEXT
```

```
3040 PRINT :
     PRINT "INSULATION R-VALUE + STRUCTURAL R-VALUE = AVERAGE R-VALUE
     ":
     RETURN
3100 CLS :
     PRINT TAB(29)"WALL":
     GOTO 3010
3200 CLS :
     PRINT TAB(29)"FLOOR":
     GOTO 3010
3999 REM   *  INPUT ERROR ROUTINE STATEMENTS   *
4000 Z = 0:
     IF DD < 200 OR DD > 10000
      THEN
       PRINT "*** ANNUAL DEGREE DAYS ARE FROM 200 TO 10,000 DAYS PER
       YEAR. ***":
       Z = 1:
       GOTO 4100
4005 RETURN
4010 Z = 0:
     IF CH < 200 OR CH > 3000
      THEN
       PRINT "*** COOLING HOURS ARE FROM 200 TO 3,000 HOURS PER YEAR.
        ***":
       Z = 2:
       GOTO 4100
4015 RETURN
4020 Z = 0:
     IF EH < 0.02 OR EH > 4
      THEN
       PRINT "*** ENERGY COST SHOULD BE BETWEEN $0.02 AND $4.  IF ENE
       RGY ***  *** COST IS  HIGHER CHANGE 'X$' IN STATEMENT # 2270.
             *** ":
       Z = 3:
       GOTO 4100
4025 RETURN
4030 Z = 0:
     IF EC < 0.02 OR EC > 0.1
      THEN
       PRINT "*** COST PER KWH SHOULD BE ENTERED IN THE FOLLOWING FOR
       M - - ****** $0.0##.IF COST PER KWH HAS RISEN ABOVE $0.10 PER
       KWH,    ****** CHANGE 'X$'IN STATEMENT # 2270.
                   ***":
       Z = 4:
       GOTO 4100
4035 RETURN
4040 Z = 0:
     IF (II OR PI) < 0 OR (II OR PI) > 60
      THEN
       PRINT "*** INSULATION R-VALUES CAN NOT BE NEGATIVE.  HOUSES WI
       TH    ****** INSULATION R-VALUES HIGHER THAN 60 MAY CAUSE IMPU
       RE AIR. ***":
       Z = 5:
       GOTO 4100
4045 RETURN
4050 Z = 0:
     IF I < 0 OR I > 36
      THEN
       PRINT "*** MONEY CAN NOT COST LESS THEN BEING FREE(ZERO COST).
         IF  ****** YOU ARE PAYING MORE THAN 36% IN INTEREST PER YEAR
       , CHECK ****** WITH A COMMERCIAL BANK FOR COMPETITIVE RATES.
                   ***":
       Z = 6:
       GOTO 4100
4055 RETURN
4060 Z = 0:
     IF E < 0 OR E > 20
      THEN
       PRINT "*** THIS PERCENTAGE IS AN AVERAGE. ONE YEAR'S INCREASE
       MAY   ****** BE 100% ,BUT OVER THE LIFE OF THE INSULATION THE
```

Program continued

```
       AVERAGE ****** SHOULD BE LESS  THAN 20% .
                  ***":
       Z = 7:
       GOTO 4100
4065 RETURN
4070 Z = 0:
     IF SF < 0 OR SF > 10 [ 5
      THEN
       PRINT "*** UNLESS YOU HAVE AN UNUSALLY LARGE HOUSE, YOU MAY HA
       VE     ****** MADE A MISTAKE IN CALCULATING THE SQUARE FOOTAGE
       OF THE  ****** AREA TO BE INSULATED. PLEASE CHECK YOUR CALAULA
       TION.      ***":
       Z = 8:
       GOTO 4100
4075 RETURN
4080 Z = 0:
     IF N < 0 OR N > 100
      THEN
       PRINT "*** YOU MUST HAVE FOUND SOME SUPER-INSULATION.  UNUSALL
       Y      ****** INSULATION WILL LAST FOR ABOUT THE LIFE OF THE HO
       USE      ****** (30 TO 40 YEARS).
                  *** ":
       Z = 9:
       GOTO 4100
4085 RETURN
4090 IF PB < 0 OR PB > 100
      THEN
       PRINT "*** YOUR PAYBACK HAS EXCEEDED 100 YEARS.  IF YOU WANT T
       O      ****** KNOW THE EXACT PAYBACK, CHANGE 'T$' TO 'Y$' IN ST
       ATEMENT ****** # 2270.
                  ***":
       GOTO 4100
4095 RETURN
4100 FOR X = 1 TO 3000:
      NEXT :
     RETURN
4999 REM   *  DATA FOR INSULATION TABLE  *
5000 DATA 0,2.5,3.7,6.0,7.4,8.8,11.0,12.3,15,15.9,19,19,22,22,30,30
5010 DATA 0,4.2,3.7,7.1,7.4,9.8,11,13,0,3.5,3.7,8.4,7.4,12.2,11,15.4,
     15,18.2,19,20.4
```

Program Listing 2. *ELECTCOM*

```
10 REM   * * * * * * * * * * * * * * * * * * * * * * * * *
         *            ELECTRIC ENERGY COMPARISON           *
         *  VERSION 1.1  - - - -  MARCH 8,1980  - - - -    *
         *            COPYRIGHT  (C)  1980                 *
20 REM   *               JOSEPH H. HART                    *
         *          2312 THOUSAND OAKS DR.                 *
         *            RICHMOND VA. 23229                   *
30 REM   * * * * * * * * * * * * * * * * * * * * * * * * *
40 CLS :
   PRINT :
   PRINT :
   PRINT :
   PRINT CHR$(23); TAB(4) STRING$(25,"* ");"     *
       *    *     ELECTRIC ENERGY    *    *     COMPARISON       *
     *                          *"
50 DEFINT A,B,C,D,F,G,H,J,M,N,O,S,X,Y:
   PRINT TAB(4) STRING$(25,"* "):
   FOR X = 1 TO 1000:
    NEXT
60 PRINT :
   PRINT :
   INPUT "DO YOU WISH INSTRUCTIONS";G$:
   IF LEFT$(G$,1) = "N"
```

```
    THEN
     1000
 69 REM    *  INSTRUCTIONS  *
 70 CLS :
    PRINT "        THIS PROGRAM ENABLES YOU TO COMPARE YOUR KWH ENERG
    Y USE FOR ONE YEAR WITH ANOTHER YEAR'S USE.  THIS WILL TELL YOU
    IF YOUR USAGE OF ELECTRICAL ENERGY HAS INCREASED OR DECREASED.
    THE PROGRAM WILL SEPARATE YOUR MONTHLY USAGE"
 80 PRINT "INTO ' BASE USE ' (LIGHTS, COOKING, RADIO, TV, ETC.)' HEA
    TING USE ' AND ' COOLING USE ' FOR EACH YEAR.  THIS METHOD ENABL
    ES YOU TO SEE WHERE YOU ARE SAVING OR USING MORE ELECTRICAL ENER
    GY.":
    PRINT @990,"(HIT ENTER TO CONTINUE";:
    INPUT G1$
 90 CLS :
    PRINT :
    PRINT "YOU WILL NEED THE FOLLOWING DATA FOR THE INPUT:
    HEATING DEGREE DAYS FOR YOUR AREA FOR EACH YEAR
         TO BE COMPARED          COOLING DEGREE DAYS FOR YOUR AREA F
    OR EACH YEAR"
100 PRINT "                        TO BE COMPARED          MONTHLY KWH
    USE FOR TWO COMPLETE YEARS"
110 PRINT :
    PRINT "      TOTAL HEATING AND COOLING DEGREE DAYS FOR A YEARCAN
    USUALLY BE OBTAINED FROM YOUR LOCAL NATIONAL WEATHER BUREAU."
120 PRINT "      THE MONTHLY ELECTRICITY USAGE (KWH) CAN BE OBTAINEDE
    ITHER FROM YOUR MONTHLY ELECTRICITY BILLS OR FROM YOUR LOCAL    E
    LECTRIC POWER COMPANY."
130 PRINT @980,"NOW YOU ARE READY TO CONTINUE (HIT ENTER)";:
    INPUT G1$:
    GOTO 1000
999 REM    *  DATA INPUT ROUTINE  *
1000 CLS :
    CLEAR 2000:
    INPUT "ENTER YOUR MONTHLY KWH USAGE FOR YEAR #-1, STARTING WITH
        JANUARY - - - - - ";J1:
    IF J1 > 5000 GOSUB 5000:
    IF Y < > 1 GOTO 1000
1010 INPUT "    FEBRUARY - - - - -";F1:
    IF F1 > 5000 GOSUB 5000:
    IF Y < > 1 GOTO 1010
1020 INPUT "    MARCH - - - - - - - ";M1:
    IF M1 > 5000 GOSUB 5000:
    IF Y < > 1 GOTO 1020
1030 INPUT "    APRIL - - - - - - - ";A1:
    IF A1 > 5000 GOSUB 5000:
    IF Y < > 1 GOTO 1030
1040 INPUT "    MAY - - - - - - - - ";MA:
    IF MA > 5000 GOSUB 5000:
    IF Y < > 1 GOTO 1040
1050 INPUT "    JUNE - - - - - - - -";JA:
    IF JA > 5000 GOSUB 5000:
    IF Y < > 1 GOTO 1050
1060 INPUT "    JULY - - - - - - - -";JB:
    IF JB > 5000 GOSUB 5000:
    IF Y < > 1 GOTO 1060
1070 INPUT "    AUGUST - - - - - - -";AB:
    IF AB > 5000 GOSUB 5000:
    IF Y < > 1 GOTO 1070
1080 INPUT "    SEPTEMBER - - - - ";S1:
    IF S1 > 5000 GOSUB 5000:
    IF Y < > 1 GOTO 1080
1090 INPUT "    OCTOBER - - - - - ";O1:
    IF O1 > 5000 GOSUB 5000:
    IF Y < > 1 GOTO 1090
1100 INPUT "    NOVEMBER - - - - -";N1:
    IF N1 > 5000 GOSUB 5000:
    IF Y < > 1 GOTO 1100
1110 INPUT "    DECEMBER - - - - -";D1:
    IF D1 > 5000 GOSUB 5000:
```

Program continued

```
     IF Y < > 1 GOTO 1110
1120 INPUT "ENTER HEATING DEGREE DAYS FOR YEAR #-1";DC:
     IF DC > 10000 OR DC < 200 GOSUB 5010:
     IF Y < > 1 GOTO 1120
1130 INPUT "ENTER COOLING DEGREE DAYS FOR YEAR #-1";C3:
     IF C3 > 3000 OR C3 < 200 GOSUB 5020:
     IF Y < > 1 GOTO 1130
1140 CLS :
     INPUT "ENTER YOUR MONTHLY KWH USAGE FOR YEAR #-2, STARTING WITH
        JANUARY - - - - - - ";J2:
     IF J2 > 5000 GOSUB 5000:
     IF Y < > 1 GOTO 1140
1150 INPUT "     FEBRUARY - - - - - -";F2:
     IF F2 > 5000 GOSUB 5000:
     IF Y < > 1 GOTO 1150
1160 INPUT "     MARCH - - - - - - - ";M2:
     IF M2 > 5000 GOSUB 5000:
     IF Y < > 1 GOTO 1160
1170 INPUT "     APRIL - - - - - - - ";A2:
     IF A2 > 5000 GOSUB 5000:
     IF Y < > 1 GOTO 1170
1180 INPUT "     MAY - - - - - - - - ";MB:
     IF MB > 5000 GOSUB 5000:
     IF Y < > 1 GOTO 1180
1190 INPUT "     JUNE - - - - - - - -";JC:
     IF JC > 5000 GOSUB 5000:
     IF Y < > 1 GOTO 1190
1200 INPUT "     JULY - - - - - - - -";JD:
     IF JD > 5000 GOSUB 5000:
     IF Y < > 1 GOTO 1200
1210 INPUT "     AUGUST - - - - - - -";AC:
     IF AC > 5000 GOSUB 5000:
     IF Y < > 1 GOTO 1210
1220 INPUT "     SEPTEMBER - - - - - ";S2:
     IF S2 > 5000 GOSUB 5000:
     IF Y < > 1 GOTO 1220
1230 INPUT "     OCTOBER - - - - - - ";O2:
     IF O2 > 5000 GOSUB 5000:
     IF Y < > 1 GOTO 1230
1240 INPUT "     NOVEMBER - - - - - -";N2:
     IF N2 > 5000 GOSUB 5000:
     IF Y < > 1 GOTO 1240
1250 INPUT "     DECEMBER - - - - - -";D2:
     IF D2 > 5000 GOSUB 5000:
     IF Y < > 1 GOTO 1250
1260 INPUT "ENTER HEATING DEGREE DAYS FOR YEAR #-2";DD:
     IF DD > 10000 OR DD < 200 GOSUB 5010:
     IF Y < > 1 GOTO 1260
1270 INPUT "ENTER COOLING DEGREE DAYS FOR YEAR #-2";C4:
     IF C4 > 3000 OR C4 < 200 GOSUB 5020:
     IF Y < > 1 GOTO 1270
1280 B1 = (A1 + MA + O1) * 1.67 * 1.6
1290 C1 = (MA + JA + JB + AB + S1) - (A1 + MA + O1)
1300 H1 = (O1 + N1 + D1 + J1 + F1 + M1 + A1) - ((A1 + MA + O1)
     * 1.67)
1310 X$ = "###,###":
     CLS :
     PRINT TAB(16)"ANALYSIS BASED ON ACTUAL USAGE":
     PRINT STRING$(63,"*")
1320 PRINT TAB(9)"FIRST YEAR"; TAB(30)"*"; TAB(39)"SECOND YEAR":
     PRINT TAB(30)"*"
1330 PRINT "JAN"; TAB(5) USING X$;J1;:
     PRINT TAB(15)"JULY"; TAB(21) USING X$;JB;:
     PRINT TAB(30)"*"; TAB(33)"JAN"; TAB(39) USING X$;J2;:
     PRINT TAB(50)"JULY"; TAB(54) USING X$;JD
1340 PRINT "FEB"; TAB(5) USING X$;F1;:
     PRINT TAB(15)"AUG"; TAB(21) USING X$;AB;:
     PRINT TAB(30)"*"; TAB(33)"FEB"; TAB(39) USING X$;F2;:
     PRINT TAB(50)"AUG"; TAB(54) USING X$;AC
1350 PRINT "MARCH"; TAB(5) USING X$;M1;:
```

```
     PRINT TAB(15)"SEPT"; TAB(21) USING X$;S1;:
     PRINT TAB(30)"*"; TAB(33)"MARCH"; TAB(39) USING X$;M2;:
     PRINT TAB(50)"SEPT"; TAB(54) USING X$;S2
1360 PRINT "APRIL"; TAB(5) USING X$;A1;:
     PRINT TAB(15)"OCT"; TAB(21) USING X$;O1;:
     PRINT TAB(30)"*"; TAB(33)"APRIL"; TAB(39) USING X$;A2;:
     PRINT TAB(50)"OCT"; TAB(54) USING X$;O2
1370 PRINT "MAY"; TAB(5) USING X$;MA;:
     PRINT TAB(15)"NOV"; TAB(21) USING X$;N1;:
     PRINT TAB(30)"*"; TAB(33)"MAY"; TAB(39) USING X$;MB;:
     PRINT TAB(50)"NOV"; TAB(54) USING X$;N2
1380 PRINT "JUNE"; TAB(5) USING X$;JA;:
     PRINT TAB(15)"DEC"; TAB(21) USING X$;D1;:
     PRINT TAB(30)"*"; TAB(33)"JUNE"; TAB(39) USING X$;JC;:
     PRINT TAB(50)"DEC"; TAB(54) USING X$;D2
1390 PRINT "HEATING DEGREE DAYS   ";DC; TAB(30)"*"; TAB(32)"HEATING D
     EGREE DAYS   ";DD
1400 PRINT "COOLING DEGREE DAYS   ";C3; TAB(30)"*"; TAB(32)"COOLING D
     EGREE DAYS   ";C4
1410 PRINT @990,"(HIT ENTER FOR RESULTS)";:
     INPUT G1$
1420 B2 = (A2 + MB + O2) * 1.67 * 1.6
1430 C2 = (MB + JC + JD + AC + S2) - (A2 + MB + O2)
1440 H2 = (O2 + N2 + D2 + J2 + F2 + M2 + A2) - ((A2 + MB + O2)
     * 1.67)
1450 BS = (B1 - B2)
1460 CS = ((C4 / C3) * C1) - C2
1470 HS = ((DD / DC) * H1) - H2
1480 CLS :
     PRINT :
     PRINT TAB(19)"ESTIMATED  ANNUAL  USE":
     PRINT STRING$(63,"-"):
     PRINT TAB(17)"YEAR #-1"; TAB(30)"YEAR #-2"; TAB(49)"SAVINGS"
1490 PRINT "BASE"; TAB(18) USING X$;B1;:
     PRINT TAB(31) USING X$;B2;:
     PRINT TAB(50) USING X$;BS
1500 PRINT "COOLING"; TAB(18) USING X$;C1;:
     PRINT TAB(31) USING X$;C2;:
     PRINT TAB(50) USING X$;CS
1505 PRINT "HEATING"; TAB(18) USING X$;H1;:
     PRINT TAB(31) USING X$;H2;:
     PRINT TAB(50) USING X$;HS
1510 PRINT TAB(17)"---------"; TAB(30)"---------"; TAB(49)"---------"
1520 T1 = (B1 + C1 + H1):
     T2 = (B2 + C2 + H2):
     TS = (BS + CS + HS)
1530 PRINT TAB(9)"TOTAL"; TAB(18) USING X$;T1;:
     PRINT TAB(31) USING X$;T2;:
     PRINT TAB(50) USING X$;TS
1540 V$ = "DECREASE":
     IF TS < 0
      THEN
       V$ = "AN INCREASE":
       TS = ABS(TS)
1550 PRINT :
     PRINT :
     PRINT "  YOU HAVE ";V$;" OF "; USING X$;TS;:
     PRINT " KWH FOR YEAR #-2 COMPARED TO YEAR #-1 AFTER ADJUSTING FO
     R BOTH HEATING AND COOLING DEGREE DAYS."
1560 PRINT @980,"FOR ANOTHER COMPARISON ENTER '1'";:
     INPUT G:
     IF G = 1 GOTO 1000:
      ELSE
       END
5000 Y = 2:
     PRINT "*** YOU HAVE ENTERED AN EXCESSIVE AMOUNT OF ENERGY ******
      USE.  IF YOUR ENTRY IS CORRECT ENTER A '1'.     ***";:
     INPUT Y:
     RETURN
5010 Y = 2:
     INPUT "** HEATING DEGREE DAYS ARE USUALLY FROM 200 TO 10,000 DEG
```

Program continued

```
     REE **** DAYS PER YEAR.  YOU HAVE ENTERED A NUMBER OUTSIDE THESE
        ** ** LIMITS.  IF YOUR ENTRY IS CORRECT ENTER A '1'.
        **";Y:
     RETURN
5020 Y = 2:
     INPUT "** COOLING DEGREE DAYS ARE USUALLY FROM 200 TO 3,000 DEGR
     EE **  ** DAYS PER YEAR.  IF YOUR ENTRY IS CORRECT ENTER A '1'.
        **";Y:
     RETURN
5030 END
```

INTERFACE

Listen to Your Keyboard

A Deluxe Expansion Interface

Interfacing the TRS-80
to the Heath H14 Printer

Listen to Your Keyboard

by Allan J. Domuret

Radio Shack is marketing a software debounce program to cure unintended multiple character generation from the keyboard. For those of you who are unfamiliar with the problem, keyboard bounce is caused by the mechanical opening and closing of keyboard switches, which results in multiple character outputs to the computer. The bounce problem can be severe if the keyboard contacts become dirty or if you have nervous fingers. Bounce can be overcome with either software or hardware, but Radio Shack neglected both, with one exception that will be discussed in the following paragraphs. Radio Shack's software fix is on the market. If you haven't already purchased it, here is my version, free. Just load it in with the Radio Shack Editor/Assembler.

In fact, I believe my debounce program is superior to Radio Shack's because mine includes generation of keyboard audio feedback so that you can hear every keystroke, accidental multiple keystrokes, and missed keystrokes, with only some minor, and optional, modifications to your cassette recorder. The audio feedback supplements the debounce software by contributing to the reduction of typing errors. As an added bonus, some cassette recorder modifications, which will allow for DEBNC audio feedback and also improve the performance of your recorder, are included.

The DEBNC program sends keyboard audio signals to the cassette recorder without activating the cassette operating relay with every keystroke. This design prevents beating the relay to death while typing, and it also keeps DEBNC from interfering with CLOAD and CSAVE functions. However, you have to manually turn on your recorder in order to hear the audio feedback. This provides a built-in safety feature, which should prevent accidental erasure of tapes left in the recorder.

Keyboard Bounce: Its Causes and Cures

As mentioned above, keyboard bounce is caused by the mechanical opening and closing of keyboard switch contacts. Figure 1 explains what actually happens every time a key is pressed. In the TRS-80, all eight data lines are held at logic zero while ROM software scans the keyboard for a keystroke. When you press a key, a logic 1 is output to the appropriate data line, which is then detected and decoded by ROM software. (The details of how ROM scans and decodes the keyboard are beyond the scope of this chapter, but for those who are interested, I recommend an excellent book by Titus, Rony, Larsen, and Titus called *8080/8085 Software Design*, published by Howard W. Sams & Co., 1978. As a relative newcomer to the

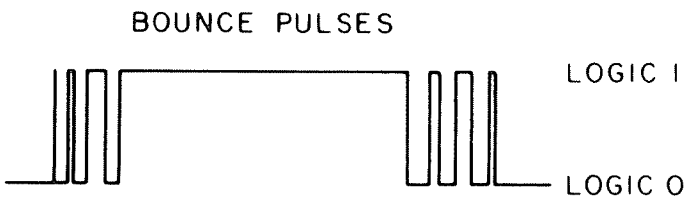


Figure 1. *Leading and trailing pulses when a key is pressed*

field of microcomputers and machine-language programming, I found this book extremely informative and easy to read, even though it is oriented to the 8080/8085 CPUs. Keyboard scanning and debounce routines are covered in chapter 7.)

Figure 1 shows the generation of a series of random pulses when a keyboard switch is initially closed. As the key is held down for a few milliseconds, the pulsations even out as the switch contacts settle down against each other. When the key is released, another series of random pulses is generated as the switch contacts separate. As a result of this switch bounce, a collection of logic 1s is sent to the data lines, and, depending on the severity of the bounce, ROM sometimes interprets these bounce pulses as multiple keystrokes rather than only one keystroke—hence, the multiple character problem. Ideally, if we could send a single pulse to ROM as shown in Figure 2, a single keystroke would be properly decoded by ROM and the multiple-character generation problem would be eliminated.

An inspection of the TRS-80 keyboard switches will help clarify the cause of keyboard bounce. Gently pry up the space bar at its center (the space bar is easier to get at than the other keys) with a plastic lever such as a thin comb. Don't use a metal pry, such as a screwdriver, or you will nick the plastic. Now, watch the exposed metal-switch contacts while you press down the square key holder. It should be fairly obvious from observing the action of these contacts that there is some inherent spring or bounce in them. Since the space bar is loose, leave it off because later we'll see how to clean all the keyboard contacts.

In order to eliminate the bounce problem, it is necessary to smooth out the leading and trailing pulses as illustrated in Figure 1, to obtain a reasonably continuous output as shown in Figure 2. Hardware such as an alternating

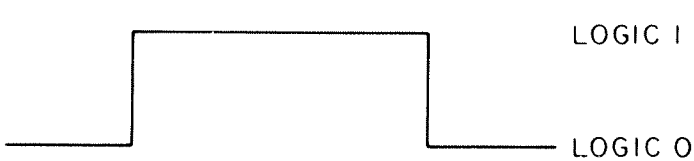


Figure 2. *Continuous output*

current rectifier circuit, can be employed to filter these pulses into a smooth output pulse, but the focus here is on software, so we won't be getting into hardware design.

The leading and trailing pulses rarely last longer than a few milliseconds, so if ROM can be convinced to ignore the first and last few milliseconds of keyboard output, it could direct its processing efforts to the center or flat part of the keyboard output pulse. The solution, then, is to tell ROM to ignore the first and last ten milliseconds or so of keyboard output, thereby solving the bounce problem. This is what DEBNC does.

TRS-80 Debounce Software

The first column in the Program Listing is the memory location for a 16K system. Note that the program resides in upper memory. For 32K or 48K systems, the ORG (ORiGinate) instruction on the top line should be adjusted to BFBCH (BFBC hex, which corresponds to 49084 decimal) or FFBCH (FFBC hex, which corresponds to 65468 decimal). For the accompanying 16K program, the 7FBCH address corresponds to 32700 decimal. These decimal addresses correspond to MEMORY SIZE? as requested by ROM when the computer is powered up; keep them handy.

Column two is hexadecimal machine language, which is automatically generated by the TRS-80 Editor/Assembler. Column three represents line numbers to ease programming and editing. Column four is the label field used to simplify addressing and branching in the program. Columns five and six are the familiar Z-80 mnemonics. Note that the labels in column four correspond to addresses referenced in column six. For instance, subroutine DELAY in column four corresponds to memory location 7FE0 hex and is referenced by the instruction CALL DELAY at memory location 7FCD hex.

Programming with the TRS-80 Editor/Assembler only requires typing in the information in columns four through six. The assembler automatically generates the line numbers and computes the memory location for the line (column three) as well as the memory location represented by the label (column four).

Debounce Relay

DEBNC keeps an eye on the keyboard, which, in its quiescent (idle) state, outputs a continuous stream of logic 0s on all data lines. If the instructions at lines 140 through 160 detect only zeros from keyboard, scanning the keyboard continues until something more interesting is detected. When a key is pressed, a logic 1 is put onto one of the data lines and lines 140 through 160 immediately recognize this different-from-zero output. Before the CPU is allowed to process this non-zero keyboard output, the debounce software introduces a short time delay of a few milliseconds to allow the

keyboard switch bounce to settle down. It is during this short delay period that the program generates an audio tone and sends it to the cassette output port. After all, why not let the computer do something useful while it is killing time?

If you study the program closely, you will note that no time delay is provided to compensate for keyboard bounce upon key release, because ROM already contains a short delay to do this. (Those of you who have a monitor such as the Small System Software RSM-1S can see the ROM CALL for this key-release time delay at memory location 044F hex. The actual delay is a subroutine at memory location 0060 hex.)

I don't know why Radio Shack designers went only half way by providing for debounce upon key release and not upon initial key press. At any rate, this is the exception I mentioned in the first paragraph.

As for the audio output, the DEBNC DELAY subroutine simply calls up the save-memory-to-cassette software in ROM and outputs a series of pulses to the cassette port. The pulses consist of alternating sync pulses used in all cassette recordings, interspersed with logic 1s (FF hex in lines 390 and 410). These pulses are sent out to the cassette as if the computer intended to record them. One concern in developing the program, however, was to keep the recorder in a normally off condition to prevent accidental tape erasures, while still preventing the computer from turning on the cassette-controlling relay every time it output a tone in response to each keyboard keystroke. This is accomplished by modifying the ROM CSAVE subroutine in DEBNC lines 340 through 380.

The cassette relay-turn-on override takes place in line 360: To turn on the motor for recording, ROM software would normally LD HL,FF04H, but instead we simply LD HL,FF00H to prevent the cassette from being turned on while still allowing the audio output to go to the cassette port. Without this feature, the ROM software, if it had its own way, would turn on the cassette every time a keystroke was output to the cassette port, and by now most TRS-80 owners are aware that such abuse of the cassette-control relay would send the relay to an early grave. Now, how do we get the cassette recorder to cooperate and give us the audio output from software? There are several options.

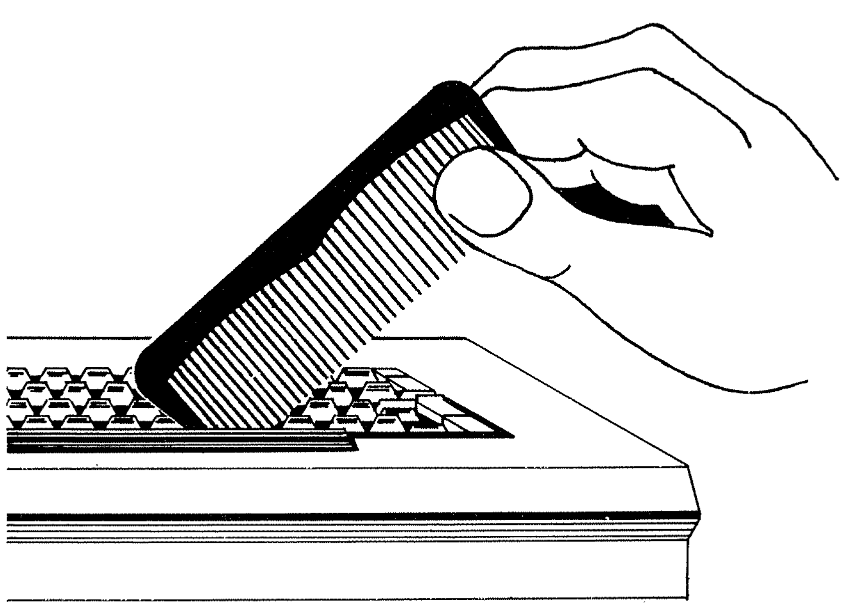
The easiest way to get the audio out of the cassette is to connect a small 3.2-Ohm speaker to a miniature phone jack and plug it into the ear output jack on the side of the recorder (see the cassette recorder modifications section for an alternative, and preferred, method). Next, it is necessary to get manual control of the cassette recorder by either pulling the remote plug from the side of the recorder or installing an override switch of the type described by Frank B. Rowlett, Jr., in *Microcomputing*, January 1979, p. 54 (for an alternative method, see the recorder mod section).

With the recorder now enabled, raise the tape cover by pressing the EJECT lever on the recorder. Then in the upper-left corner of the tape cavity you will find an "erase-protect" lever that protrudes when you attempt to depress the RECORD lever. Hold this erase-protect lever in while simultaneously depressing the RECORD and PLAY levers as you would in preparing a recording. Manually holding in this erase-protect lever enables the red RECORD lever to be depressed. This activates the cassette amplifier and allows the audio from the computer to enter the amplifier via the cassette aux input.

By now you probably have noticed one glitch. This procedure keeps the cassette motor running continuously while DEBNC is used in this mode. If you spend hours typing a BASIC program into the computer using DEBNC with its audio feature, your cassette motor will run for these same hours. You have several options:

1. Let the motor run. It has a long life, and you really won't hurt it.
2. Install a motor turn-off switch to deactivate the motor without defeating the cassette amplifier. This, too, is covered in the recorder mod section.
3. Ignore the audio output. The debounce program will still use the audio output subroutine to generate the necessary debounce time delay, but you just won't hear it and it won't hurt anything.
4. Feed the audio tone to a separate amplifier.

Notice the built-in safety feature of this design. There is no way to activate



the cassette recorder with DEBNC and accidentally erase a valuable cassette tape. Of course, it is possible to leave a tape in the recorder to enable activation of the RECORD/PLAY levers, but the danger of doing this, I believe, is low. By now, most computerists have developed good tape-handling practices so as to avoid such accidents.

Perhaps it would be worthwhile to mention the purpose of lines 170 through 200 in DEBNC. Without these program steps, the TRS-80 keyboard would output what would sound like a continuous audio output for as long as a key remains depressed. As a key is held down, keyboard scanning continues and an audio tone would be output on every scan cycle for as long as the key is held down. Lines 170 through 200 determine if the keyboard output is the same as it was in the last scan cycle. If so, it skips the tone generating delay. If a keyboard output that is different from the last scan output is detected, then the delay is permitted. This technique still preserves the debounce feature.

Lines 450 and 460 in DEBNC are used to gain control of the keyboard scan routine. In normal operation, the keyboard memory scan routine vector is stored in memory locations 4016H and 4017H. When ROM wants to scan the keyboard, it calls the contents of memory locations 4016H and 4017H and finds the ROM scan routine at memory location 03E3H. To gain control of the keyboard scan routine, it is necessary to change the contents of 4016H and 4017H so that the jump will be to DEBNC at memory 7FBCH instead of to 03E3H. This is what lines 450 and 460 in DEBNC do.

If for some reason it becomes necessary to RESET the computer while DEBNC is working, ROM will regain control of the keyboard scan and DEBNC will be defeated. If this happens, it will be necessary to reload DEBNC. This should be no problem because DEBNC only takes a few seconds to load.

It might occur to you as it did to me to POKE the DEBNC start address into 4016H and 4017H. It won't work. Any attempt to change the keyboard scan vector located in 4016H and 4017H while ROM is busy scanning will crash the system. This will require turning the computer off and back on to reset everything.

Loading and Operating Debounce

Upon RESET or initial application of power, enter the appropriate memory size when so requested by the computer. Use the addresses as provided above in the TRS-80 Debounce Software section. As an example, for a 16K system, DEBNC should originate at memory 7FBCH, which corresponds to MEMORY SIZE 32700 decimal (enter the decimal figure into the computer, not the hex number).

Next, the usual system code is entered, followed by the file name, DEBNC, to start loading. After the tape loads, hit the BREAK key, and

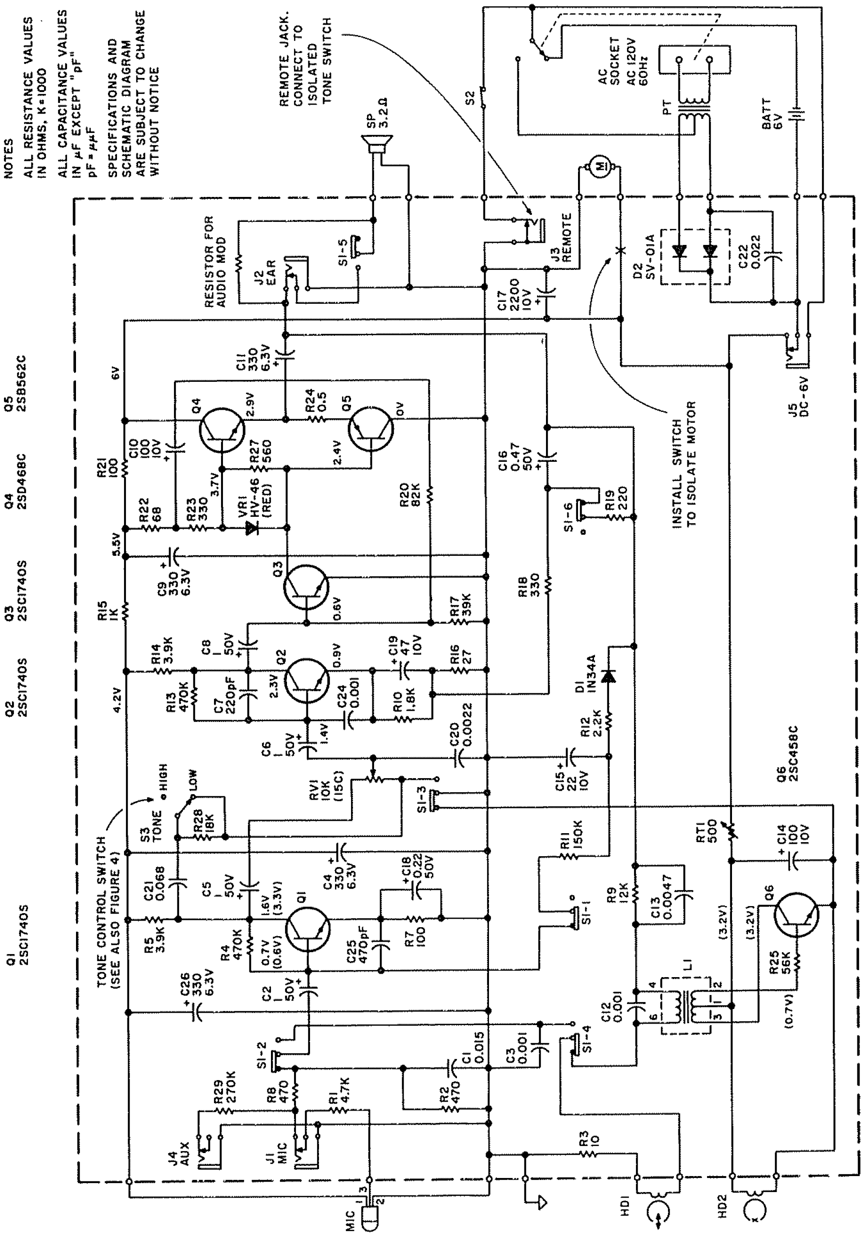


Figure 3. *CTR-41 cassette recorder schematic*
(Reproduced by written permission from Tandy Corporation.)

DEBNC is ready. This is a departure from typical system tape-loading procedures. No slash (/) key or ENTER key should be pressed because the modified keyboard vector which has been loaded into 4016H and 4017H automatically addresses DEBNC.

Finally, set up the cassette recorder as described in preceding paragraphs if audio feedback is desired. Personally, I find the audio feedback indispensable, because it eliminates many typing errors.

Cleaning Keyboard Contacts

While you are sitting there with your space bar still hanging out, use your plastic comb, or whatever, and pop off all the other key caps to expose the key contacts. Now spray all the key contacts with tuner cleaner, rubbing alcohol, or something similar.

Three cautions should be observed in this cleaning process. First, don't use a cleaner that could mar or otherwise damage your plastic keyboard. Perform a chemical reaction test using the cleaner on the bottom of your keyboard where possible melting or damage won't show. Second, don't use cotton swabs to dab liquid cleaner on the contacts. The cotton may leave small threads on the contact which could interfere with normal operation of the contacts. And third, don't put any unguents on the contacts, such as Vaseline, which is an insulator, not a conductor, and will only serve to latch onto dust, cigarette smoke particles, and so on to the extent that the contacts will become inoperative, either wholly or partly.

Of course, if your TRS-80 is new, this cleaning procedure should not be necessary, but if your keyboard has been sitting on the table uncovered for months, the cleaning will not hurt. As a final protection, keep your keyboard covered when not in use. The debounce software should solve most of your bounce problems, and proper care and cleaning of the key contacts will also help, even without the debounce software.

TRS-80 Cassette Recorder Modifications

A schematic of the Radio Shack CTR-41 cassette recorder, extracted from the owner's manual, is provided in Figure 3. Four modifications are recommended, and three of them are, in my opinion, indispensable even without the use of debounce software. These mods have been suggested in various forms by other hobbyists, most of them requiring some kind of external controlling box. Refer to both the schematic and the accompanying printed circuit board sketch (Figure 4) when making the mods.

1. *Audio Modification:* Connect a resistor (I used a 47-Ohm) across the top speaker wire and the top ear connector (J2) as shown in both figures. Different size resistors will provide different volume levels. Experiment to find a suitable volume level. Figure 4 shows where to connect this resistor on

the printed circuit board. In addition to allowing use of DEBNC audio, this resistor will also allow you to hear both CSAVE and CLOAD audio without external boxes or without the necessity of pulling plugs on the side of the recorder. With this mod, you will have no more recording surprises as a consequence of not hearing what was going into or out of the recorder. If desired, a switch can be installed in series with this resistor to defeat it.

2. *Separate Motor Control:* A switch in series with the motor as shown in the schematic will permit shutting off the motor when only the cassette amplifier is desired for DEBNC audio. This is not shown in Figure 4 because I have not installed such a switch.

3. *Computer/Manual Cassette Control:* In Figure 3, locate the tone control, S3. Isolating the switch from the circuit without disturbing R28 and C21 leaves the tone circuitry in the high mode as it should be for computer use. When properly wired, this switch can be used to get manual control of the recorder without external mods and without pulling out the remote jack. See Figure 4 for instructions as to where to cut leads on the board to isolate the tone control switch. Now run two wires from the switch to the two connectors on the remote jack as shown in Figures 3 and 4.

4. *Ground Loop Mod:* As long as your recorder is disassembled, this is a good time to do another indispensable mod. The stock Radio Shack CTR-41 recorder is notorious for generating hum via ground loops when used with

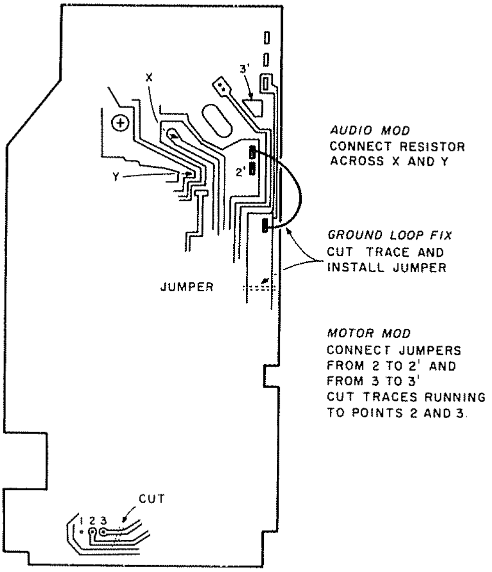


Figure 4. *CTR-41 printed circuit board.*

the TRS-80. The fix is to cut the board trace and run a jumper wire as shown in Figure 4. This fix will greatly reduce hum on computer-generated tapes and will also reduce loading problems. There are other methods for curing the ground loop problem, but this one keeps the mod inside the recorder where it belongs, out of sight. It is my understanding that newer TRS-80 recorders have some of these mods installed, especially the ground loop fix, so it may be necessary to perform only mods one and two to isolate the cassette motor while using DEBNC with audio. You will have to determine your own needs.

Program Listing. *DEBNC program symbolic list*

```
7FBC            00100           ORG     7FBCH
7FBC 213640     00110 DEBNC     LD      HL,4036H
7FBF 010138     00120           LD      BC,3801H
7FC2 1600       00130           LD      D,00
7FC4 0A         00140 CKKEY     LD      A,(BC)
7FC5 A7         00150           AND     A
7FC6 2809       00160           JR      Z,ZERO
7FC8 5F         00170           LD      E,A
7FC9 7E         00180           LD      A,(HL)
7FCA BB         00190           CP      E
7FCB 2803       00200           JR      Z,INAGN
7FCD CDE07F     00210           CALL    DELAY
7FD0 0A         00220 INAGN     LD      A,(BC)
7FD1 5F         00230 ZERO      LD      E,A
7FD2 AE         00240           XOR     (HL)
7FD3 73         00250           LD      (HL),E
7FD4 A3         00260 INCSCN    AND     E
7FD5 C2FA03     00270           JP      NZ,03FAH
7FD8 14         00280           INC     D
7FD9 2C         00290           INC     L
7FDA CB01       00300           RLC     C
7FDC F2C47F     00310           JP      P,CKKEY
7FDF C9         00320           RET
7FE0 3E00       00330 DELAY     LD      A,0
7FE2 32E437     00340           LD      (37E4H),A
7FE5 E5         00350           PUSH    HL
7FE6 2100FF     00360           LD      HL,0FF00H
7FE9 CD2102     00370           CALL    0221H
7FEC E1         00380           POP     HL
7FED 3EFF       00390           LD      A,0FFH
7FEF CD6402     00400           CALL    0264H
7FF2 3EFF       00410           LD      A,0FFH
7FF4 CD6402     00420           CALL    0264H
7FF7 CDF801     00430           CALL    01F8H
7FFA C9         00440           RET
4016            00450           ORG     4016H
4016 BC7F       00460           DEFW    DEBNC
0000            00470           END
00000 TOTAL ERRORS

CKKEY   7FC4 00140    00310
DEBNC   7FBC 00110    00460
DELAY   7FE0 00330    00210
INAGN   7FD0 00220    00200
INCSCN  7FD4 00260
ZERO    7FD1 00230    00160
```

A Deluxe Expansion Interface

by Frank Delfine

Planning to expand your TRS-80? Not sure which system is best? If you want to use your TRS-80 to do more than just write BASIC programs, you should consider the expansion interface presented here.

After reviewing the features of the Radio Shack expansion interface, I decided that it just wasn't flexible enough for the projects I had planned for my TRS-80. Since most of my computer projects involve custom-designed hardware interfaced with a microcomputer, I was looking for a way of utilizing the TRS-80 as a development tool for these projects. With this goal in mind, I designed the Deluxe Expansion Interface. Photo 1 shows the TRS-80 with the expansion box and disk drive.

Features of the Deluxe Interface

The Deluxe Expansion Interface has the following features:

1) Allows the use of up to four 5 1/4-inch minifloppy drives.

2) Provides a parallel printer port.

3) Provides a serial RS-232C communications port.

4) Connects the TRS-80 keyboard bus to an S-100 motherboard.

5) When constructed with the specified S-100 mainframe will provide 12 slots for expansion plus a 20-Amp +8-volt supply and a 4-Amp ±16-volt supply for supporting future boards in the system.

6) Provides 16K of static R/W (read/write or random access) memory for a 32K TRS-80 system, which is easily expandable to 32K for a full 48K system.

7) Provides a real-time clock.

This system forms the foundation of my general-purpose microcomputer development station. Future boards for the system include devices such as ICE packages (in circuit emulator), PROM programmers, general-purpose I/O boards, high-speed tape backup, and 8-inch floppies, all of which are nicely supported (both electrically and mechanically) by this expansion box.

System Configuration

The system is divided into four sections:

1) The S-100 mainframe with motherboard and power supplies.

2) The IPC board (interface/printer/communications): contains the buffering and data flow control circuitry necessary to interface the TRS-80 bus to the S-100 bus properly. The parallel printer port and the RS-232C serial port also reside on this board.

3) The FDC board (floppy disk controller): controls the floppy disk operations.

4) The static RAM board: contains 16K of static R/W memory. A second memory board may be added to provide a full 48K system.

Mainframe/Motherboard Description

The mainframe is manufactured by California Computer Systems and includes a 12-slot S-100 motherboard. The motherboard features active termination and a crisscross type of PC pattern to act as a sort of twisted pair line. This should tend to minimize the noise pickup and cross talk from adjacent PC board runs. The power supply provides you with an unregulated +8 V dc at 20 A as well as unregulated ±16 V dc at 4 A. In an S-100 system the power supplied to the bus connectors is always unregulated, therefore, the regulation must be handled on each card in the system by employing three terminal regulators. This does away with the need for a massive heat sinking and regulating stage in the back of the enclosure and insures that the correct voltage is supplied to each board, since any voltage drops along the motherboard are on the input side of the regulator stage. This is an important consideration when you have several boards, each capable of drawing a few Amps, plugged into a motherboard that is 12 slots long. I mention this in case some of you would like to build a custom enclosure and were thinking of using a regulated supply.

In addition to the enclosure, supplies, and motherboard, the mainframe contains a cooling fan and cutouts on the back panel for various connectors. The mainframe should be operated with the cover in place for the best results from the cooling fan.

IPC Board Operation and Description

The IPC board buffers the signals from the TRS-80 keyboard connector and routes them to the S-100 bus for use by any other card plugged into the motherboard. In addition to providing a buffering operation, the address bus is monitored and only allows the data bus buffers to become enabled when a device which is not in the keyboard is accessed. The addresses of concern here are those in the memory-mapped I/O section of the TRS-80 memory map, as well as all addresses above 7FFFH, the end of the 16K keyboard system. The data buffers must also be enabled for all I/O operations with the exception of the cassette port located in the keyboard at port 255. Since this is not in the expansion box, the data buffers must be disabled during any cassette operations.

Referring to Figure 1, ICs 1 and 2 are the address buffers. They are enabled at all times, since in this system we are allowing only the Z-80 in the keyboard to ever have control of the address bus. If any DMA (direct

memory access) operations or multiprocessor designs were to be used in the expansion box, the address buffers would have to be controlled accordingly. ICs 3 and 4 are the data bus buffers. Two buffers are used to create a single bi-directional bus. IC3 allows data to flow from the expansion box to the keyboard. IC4 allows data to flow from the keyboard to the expansion box. To understand how these buffers are controlled, we start at address decoder IC9A and IC9B. These gates, along with IC6A, are set up to detect any address whose high-order byte is equal to 37H. The address map for the TRS-80 shows that this is where the disk drive, line printer, and interrupt latch (the function of the interrupt latch will be discussed later) reside. Referring again to Figure 1, we can see that this 37xxH signal is gated with AS15* in IC5D. The output of IC5D will be a low anytime an address greater than or equal to 8000H is placed on the address bus or if 37xxH is generated on the bus. The only other conditions left to detect are any I/O operations and a cassette operation. IC23, along with IC6B and IC6C, forms the cassette port decoder. Any cassette operations will force the output of IC6D high, disabling the data buffers through IC8A and IC8B. Any other I/O operation is detected by IC5C and causes IC6D's output to go low, letting the buffers operate.

The RS-232C serial port is shown in Figure 2. It is mapped into several I/O ports between E8 and EB. The function of each of these ports is listed in Table 1. IC33A and IC26, along with seven gates from IC30 and 31, form the address decoder to provide the I/O device-select pulses. These particular ports match the port addresses used in the Radio Shack expansion interface. This means that existing software drivers will work here also.

The 1488 and the 1489 ICs (ICs 28 through 32) are line driver and receiver chips which accomplish the level shifting between the UART's TTL levels and the ± levels required for the RS-232C specification. The CTS, (clear to send), DSR (data-set ready), CD (clear data), RI (right indicator), RTS (ready to send), and DTR (data terminal ready) signals are modem handshaking signals and are provided so that the port may interface with a modem to tie the TRS-80 into a time-sharing network. RD (received data) is the serial data coming into the UART, while TD (transmitted data) is the serial stream put out by the UART.

The AY-5-1013A UART used requires +5 V and −12 V. While there are other devices available (such as the AY-3-1015) which require only +5 V and are pin compatible, they tend to be more expensive. Since the line driver chips require ± 12 V, the power supply will be there anyway, so you may as well use the cheaper chip.

The next function to be discussed is the parallel-printer port. The printer is mapped at 37E8H. The printer requires eight data bits plus a strobe pulse. The CPU looks at the upper four bits of the port for the printer status. This means we need an eight-bit latched output port and a four-bit input buffer

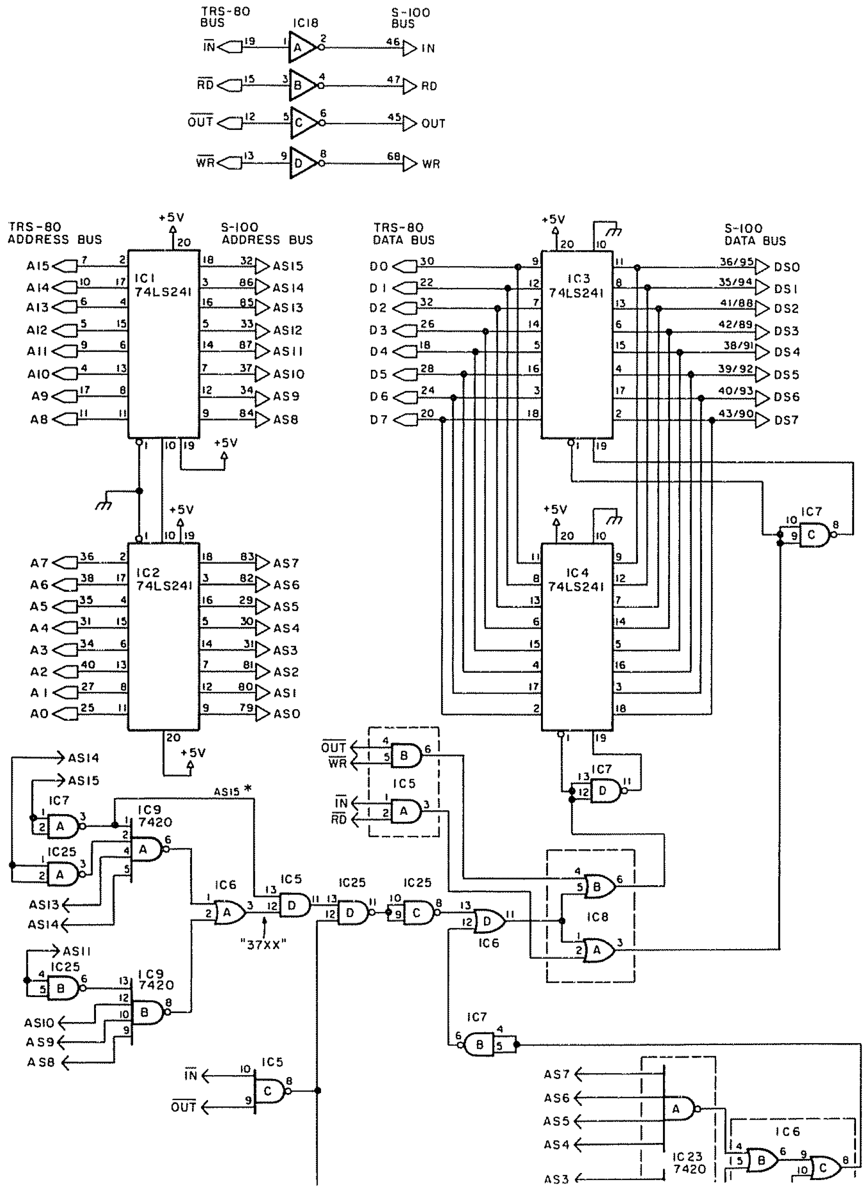


Figure 1. *Partial schematic of IPC board*

at 37E8H. IC20 serves as the output latch, while IC21 acts as the input buffer. In order to provide a suitable strobe to the printer, the 37E8H write pulse triggers the one-shot IC22 and stretches the pulse width to accommodate the printer.

The only remaining functions on the IPC Board are the buffering of the IN*, RD*, OUT*, and WR* signals by IC18 (see Figure 1), the SYSRES* signal by IC19D, and the generation of three device-select signals for use by the FDC board (see Figure 3). These signals are DISK SELECT (37ECH to 37EFH), INTERRUPT LATCH (37E0H), and DISK DRIVE SELECT (37E1H). They are developed by ICs 11, 12, and 19 and are passed to the FDC board via three unused pins on the S-100 bus (pins 13, 14, and 15). Table 2 gives a complete IC list for the IPC board.

FDC Board Operation and Description

Disk operation is controlled by the operating system software and the INS1771 controller chip (see Figure 4). This chip makes interfacing to the disk drive a relatively simple matter. As I mentioned earlier, the disk controller utilizes four memory locations (37ECH through 37EFH). This is how the CPU reads and writes status information, command information, and data to and from the controller chip. Table 3 lists the function of each of these addresses in more detail.

The disk operates on an interrupt basis. When the controller chip has some data for the processor, it brings the INTRQ pin high causing the interrupt latch (IC11) to set. The output of the interrupt latch is connected to the processor interrupt pin via the IPC board. If we examine the interrupt latch more closely, we see that in addition to the disk interrupt setting the latch, IC8B can also cause an interrupt to occur. IC8 forms the "heartbeat" interrupt latch. This is what updates the real-time clock in the operating system software. The input to IC8A is a 40-Hz pulse train derived from the 8.0 MHz clock IC1. This causes an interrupt to occur every 25 ms. The operating system uses this interrupt to update a counter in software that can be displayed as the time of day. (When the CMD"T" command is issued in Disk BASIC, it causes the interrupt to be disabled in software. This is necessary when you want to carry out cassette operations, since these are time-dependent, and any interrupts would cause a loss of data.) Since we have two devices which can cause IC11 to generate an interrupt to the processor, we need a way to inform the processor which device must be serviced. This is handled by buffer IC12. When an interrupt occurs, the processor jumps to a service routine which reads the buffer (IC12) at 37E0H. If D7 is high, the interrupt is from the clock. If D6 is high, the disk has caused the interrupt. The routine can now jump to another routine to service the device.

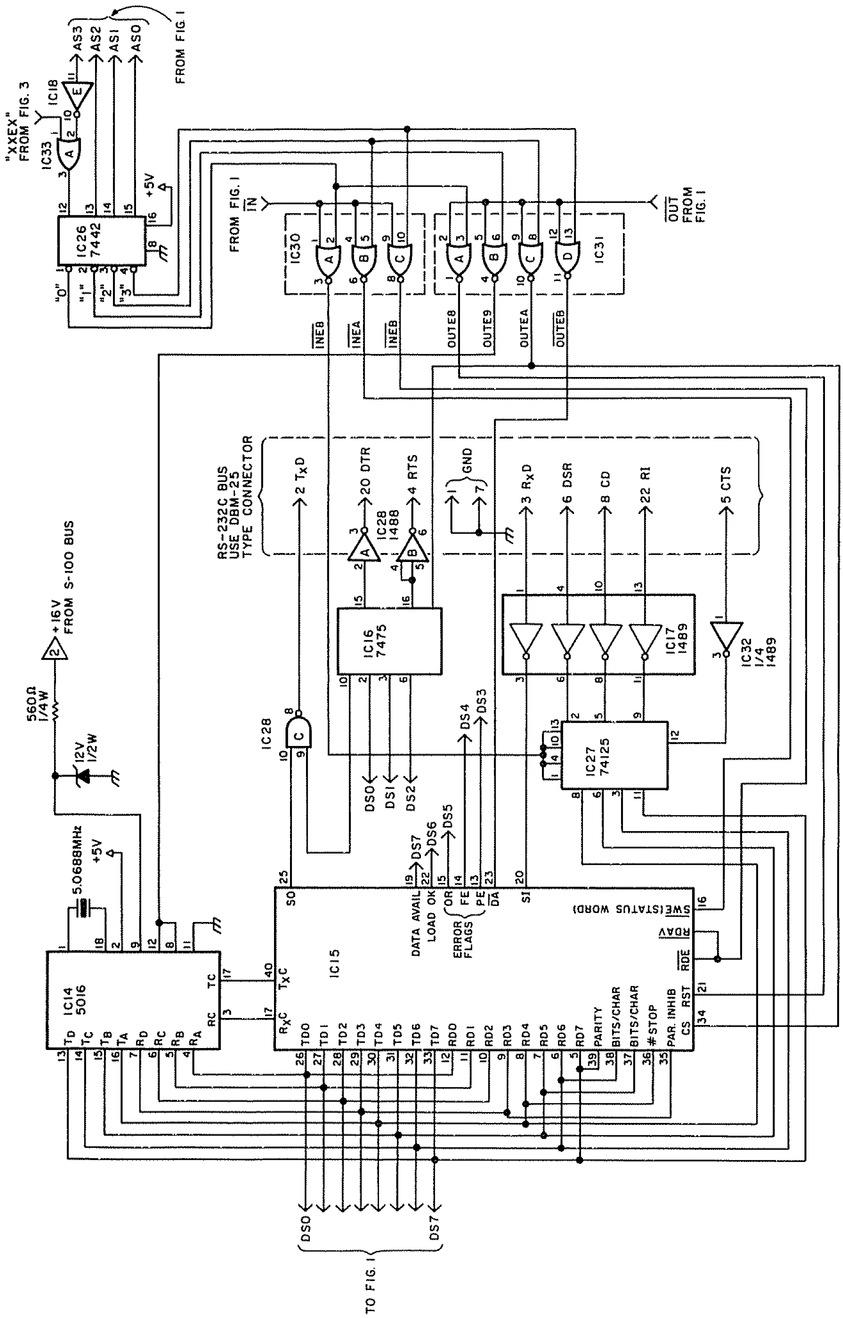


Figure 2. *RS-232C port*

When a disk operation is initiated, a drive is selected and the motor is turned on. This is done by writing a drive select word to 37E1H. This word is latched by ICs 15 and 20 and selects a drive by pulling one of the drive select lines low on the Shugart bus to the drives. The 37E1H write pulse also causes the one-shot IC13A to start the drive motor. See Table 4 for a complete IC list for the FDC board.

Before I conclude my discussion of the disk, I will discuss data separation. When data is read off the disk, it appears to the controller as a stream of clock pulses with the data contained within a certain time frame, or window, between the clock pulses. The controller chip utilizes the 1-MHz clock out of IC2B to determine the location of the data window. Since this clock is asynchronous with the data coming from the disk, the data can occur outside of the data window and be lost. To solve this problem, the chip manufacturer recommends the use of an external data separator. Such a circuit is shown in Figure 5. It can be added to the disk controller circuit of Figure 4 to provide a more reliable system. For more information on the interfacing of disk drives, I direct your attention to the references I have provided at the end of this chapter. Table 5 gives a list of ICs needed for the FDC separator.

16K Static RAM Board

Some of you might be wondering why I chose to use static instead of dynamic memory for this project. While it is true that dynamic memory does have advantages over static memory, such as lower cost and chip count, I feel reliability and ease of circuit reproduction is an issue in a project such as this. Dynamic memory utilizes a single transistor as a charge storage element for its bit cell. Since the charge will leak off after a short period of time, the array must be periodically recharged or refreshed to avoid losing data. Another thing to take care of in a dynamic system is address multiplexing. The address in a 16K dynamic RAM chip, such as the 4116, is presented to only seven pins. Since 14 address lines are required to address 16K of memory, these pins are multiplexed by using the MUX (multiplexer), RAS (row address strobe), and CAS (column address strobe) signals from the CPU. When the MUX signal is low, a RAS pulse is applied to the chip which latches the lower seven bits into the chip. A short time later, the MUX signal goes high, and a CAS pulse is applied, latching the upper seven bits into the chip. These strobes are rather fast and would have to be brought to the expansion box via the ribbon cable at the rear of the keyboard. Since the cable must be about three feet long, this could prove to be a problem as far as noise and capacitive/inductive effects are concerned. An elaborate buffering scheme in the cable and on the IPC board would need to be added. Another

consideration in dynamic systems is proper and adequate bypassing of the power supply lines at the chips themselves. The component locations as well as the values become critical in the design. This would prove to be an unworkable situation for someone trying to make a wire-wrap copy of the board. In addition to these points, I have had experience with some Radio Shack expansion interfaces where a relatively small amount of ac line noise does cause memory errors. Even an expensive line regulator does not completely eliminate the problem. For these reasons, I decided a static memory would be the better solution.

The schematic of the memory board is shown in Figure 6, and the IC list is shown in Table 6. The 16K × 8 memory itself is formed from 32 1K × 4 2114s. The address decoder (IC2) is set up so that it is only enabled when an address of 8000H to BFFFH is present. This represents a 16K block of memory. IC2 breaks this block into 16 1K segments to provide a chip select signal for each pair of 2114s. For a 48K system, you must add a second board with the decoder wired for a start address of C000H (see insert in Figure 6). As an alternative to wire-wrapping 37 ICs onto a plugboard, you can purchase an assembled S-100 16K RAM board and just plug it in. If this is a bit

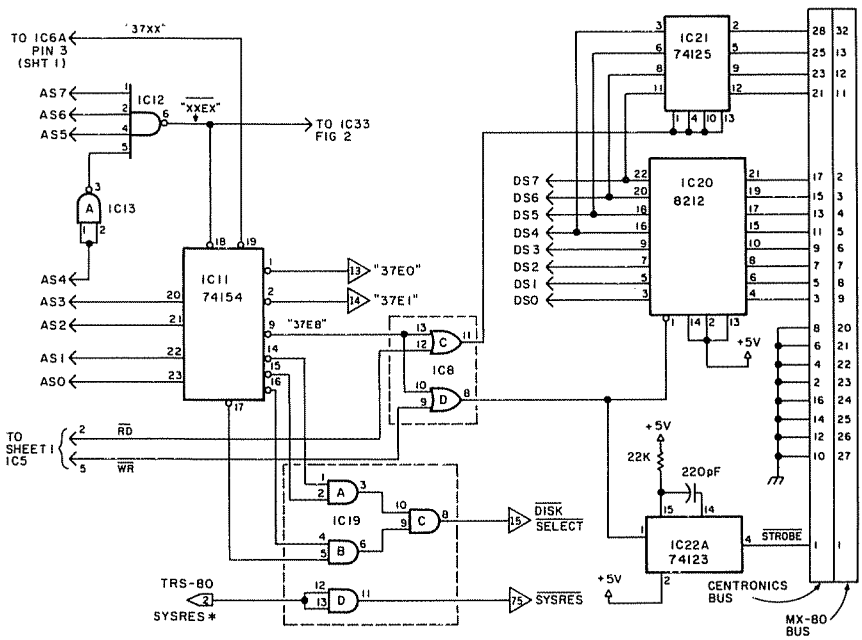


Figure 3. *Partial IPC board*

too expensive, blank PC boards are available which save you the wiring but you supply your own chips.

Before constructing the wire-wrap memory board, I tried out an old 4K RAM board from a Processor Technology SOL system. All that was required was setting the address DIP switches to start at 8000H and plugging it in. Instant 20K system! While the address, data, and read/write lines are compatible with the S-100 conventions, there are some extra signals on the S-100 bus which are not used by the TRS-80. These signals tend to be defined as active high. Since the motherboard has active termination, they will get pulled up by the bus automatically. There may be some S-100 boards which will require some signals to be jumpered for proper operation.

As a closing comment on the memory board, I would like to point out that static memories are beginning to become more densely packaged. At the time of this writing, NEC has published a preliminary spec on a 16K $\times$ 1 static memory. This is the same density as a 4116 dynamic RAM. These chips would allow the circuit of Figure 6 to be built with only 8 ICs for the memory instead of 32. These chips are not yet available and probably will be rather expensive at first, but it appears that static memory will more closely approach dynamic memory in cost effectiveness in the near future.

Construction and Troubleshooting

All of the boards are wire-wrapped on prototype plugboards. Connections between the keyboard and IPC board are made via a 36-inch length of 40-conductor ribbon cable. You will have to get a 40-pin edge connector to mate with the keyboard PC card and another connector and mate for the IPC board. I used 3M connectors on the ribbon cable, but there are many other types available. The connectors can be crimped onto the ribbon cable easily with just a 2 1/2-inch vise. Carefully line the ribbon in the connector, making sure that each wire is positioned at the center of each pin. You may have a problem if the ribbon cable has been stored on its edge. This causes the wires to get a little closer together than they should be. If this happens, gently stretch the cable at the connector end until it fits the connector. The ribbon should be left protruding from the connector about 1/2 inch. This can be trimmed off later. Once everything is aligned, press down on the connector with your fingers to get things started. Place the connector in the vise, and compress it until the two halves touch. Remove the cable from the vise, and trim off the excess cable from the connector with a knife. Take your time with these cables, since a mistake here could cost you hours in debugging time later.

I have found that the best way to get a project like this working is to build and test one section at a time. The IPC board should be the first card built. Start with the voltage regulators (Figure 7 shows the power supply for the

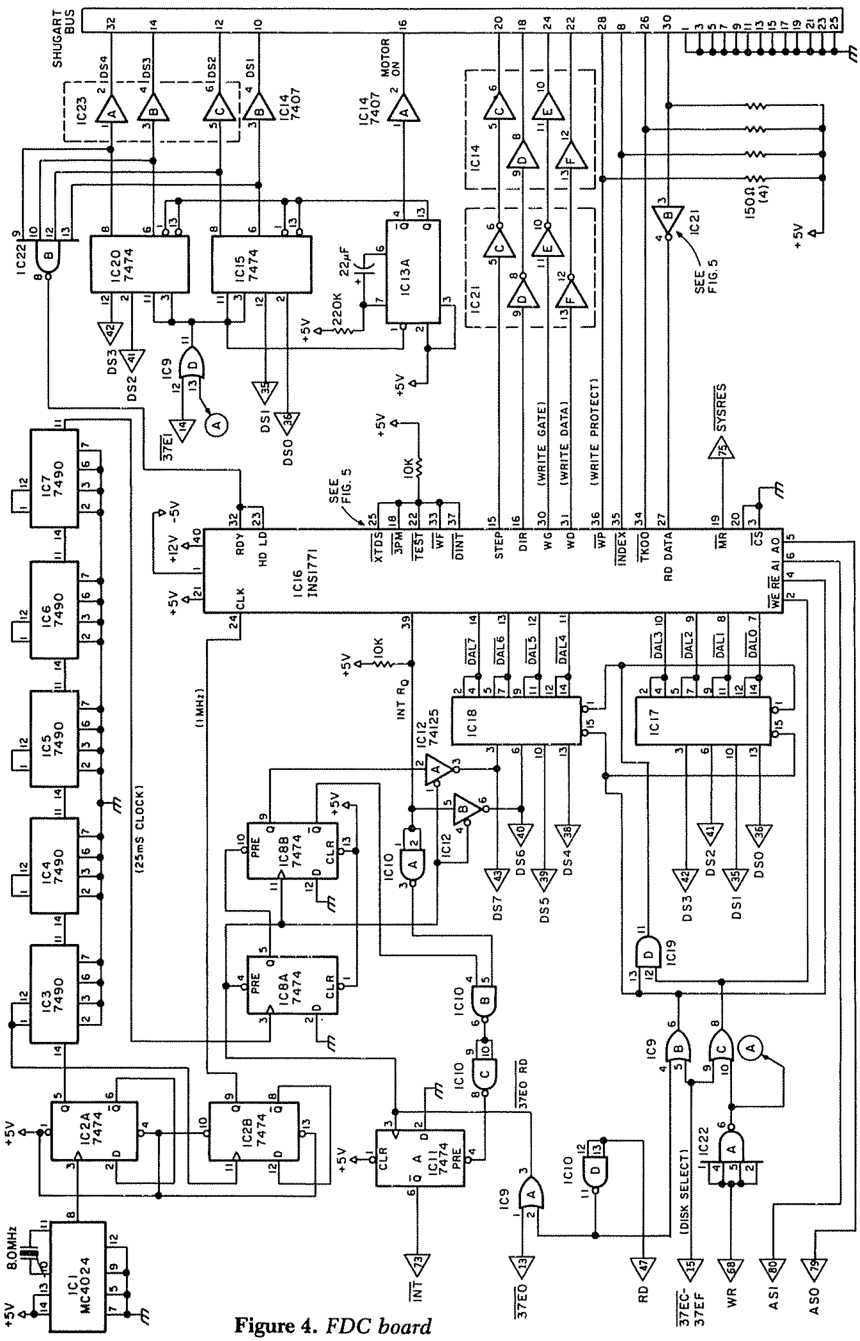


Figure 4. *FDC board*

IPC board), address and data buffers, and address device decoders. After you have these all wired up, plug in the card without any ICs, and check for proper supply voltages at the IC sockets. You may want to get an S-100 extender card to aid in making measurements on the cards while they are in the mainframe. If all looks well here, turn off the mainframe, unplug the card, and install the chips. Now plug the keyboard into the IPC board, and turn on the mainframe and the TRS-80. If the TRS-80 doesn't power up properly or locks up on you, turn everything off and check for proper cable orientation. *Note that the TRS-80 keyboard does not necessarily have the same pin orientation as the connector you may be using. Take this into consideration when you wire the connector to the IPC board.* Check also for any shorted pins on the wire-wrap connector on the IPC board, and be sure the buffers are oriented properly in their sockets. When you have the keyboard functioning properly with the IPC board connected and powered up, you are ready to run some tests. To check that the address decoders are working properly, you can write a little loop in BASIC using the PEEK and POKE commands to generate device select pulses. You should run these test loops and look at *all* the device selects with an oscilloscope. Make sure that not only the proper devices are being selected but that no other devices are inadvertently being enabled. As an example run:

```
10 PRINTPEEK(14312):GOTO10
```

and look for a series of pulses on IC8 pin 11. This is the printer status port. Check all other outputs on IC11. The data bus buffers should also be checked to see if they are enabled by the proper devices and in the proper direction. If these buffers turn on at the wrong time, they could cause the TRS-80 to hang up when powered on.

Once the device selects are working, the ports can be added. Start with the printer and add the RS-232C port (this port may be omitted entirely if you don't care to use the serial feature). The printer is connected to the IPC board via a 26-conductor ribbon cable. The particular connector used at the printer end will depend on the exact printer you are going to use. I have listed the pinouts for the standard Centronics bus, since most printers follow this convention. My system uses an Epson MX-80 which does have a different pinout, and I have listed this in addition (see Figure 3).

To use the RS-232C port, you will need a driver routine in software. There are many available for the TRS-80, so I will not present one here. If you wish to write your own driver, Table 1 will assist you in defining the control and data functions.

An oscilloscope can be used to trace the data in and out of the drivers and UART. Remember the output of the 1488 should be between ± 6 volts. The 1489s are expecting to see an input swing between some value greater than

± 3 volts. You should always have a TTL level at the input and output pins of the UART.

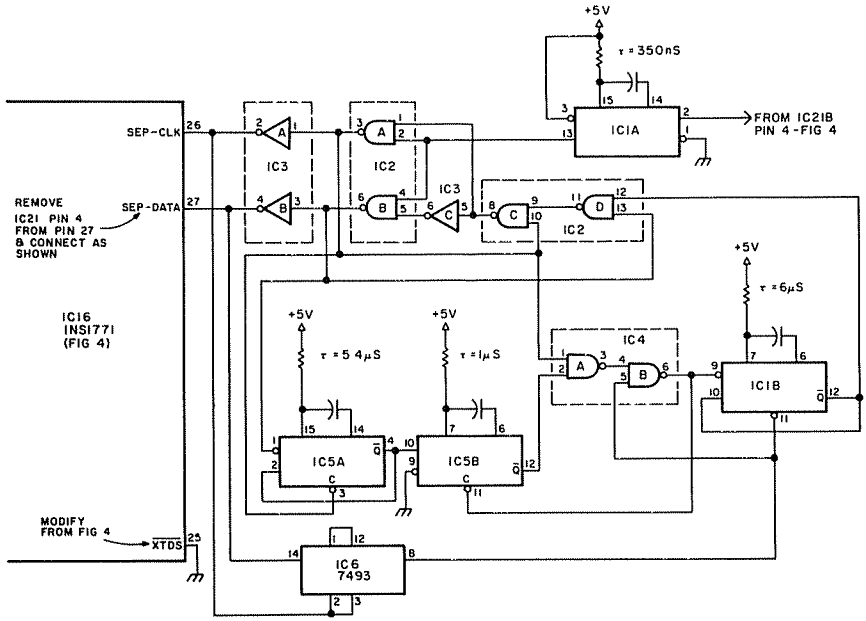


Figure 5. *Optional data separator for FDC board*

Once the IPC board is working, you can proceed to the floppy. To aid in troubleshooting, you may want to build the circuit in Figure 4 first, get the disk working, then go back and add the external data separator in Figure 5 (see Photo 2). The circuit is pretty straightforward, and a little care in the wiring should result in a working board the first time. Each time the reset button on the keyboard is pressed, the program goes out and reads the 1771's status to see if a disk is connected to the system. You should see a pulse at the 37EC address select (pin 15 on the S-100 bus—see Table 7 for a complete S-100 pinout listing) every time you reset or power the keyboard on. If all is well, you should be able to insert a system disk, press the reset button, and in a few seconds the DOS READY prompt should appear. Before trying your system disk for the first time, you should try to get a backup made just in case you run into problems at the beginning of your testing. If the system is powered up with the FDC card installed but without a disk in the drive, you will get a combination of graphics/alphanumeric characters on the screen. This is a normal situation. If you want to get to Level II BASIC from this point, hold down the BREAK key while pressing the reset button. To get into Disk BASIC, insert a disk and press the reset button. (Figure 8 shows the power supply for the FDC board.)

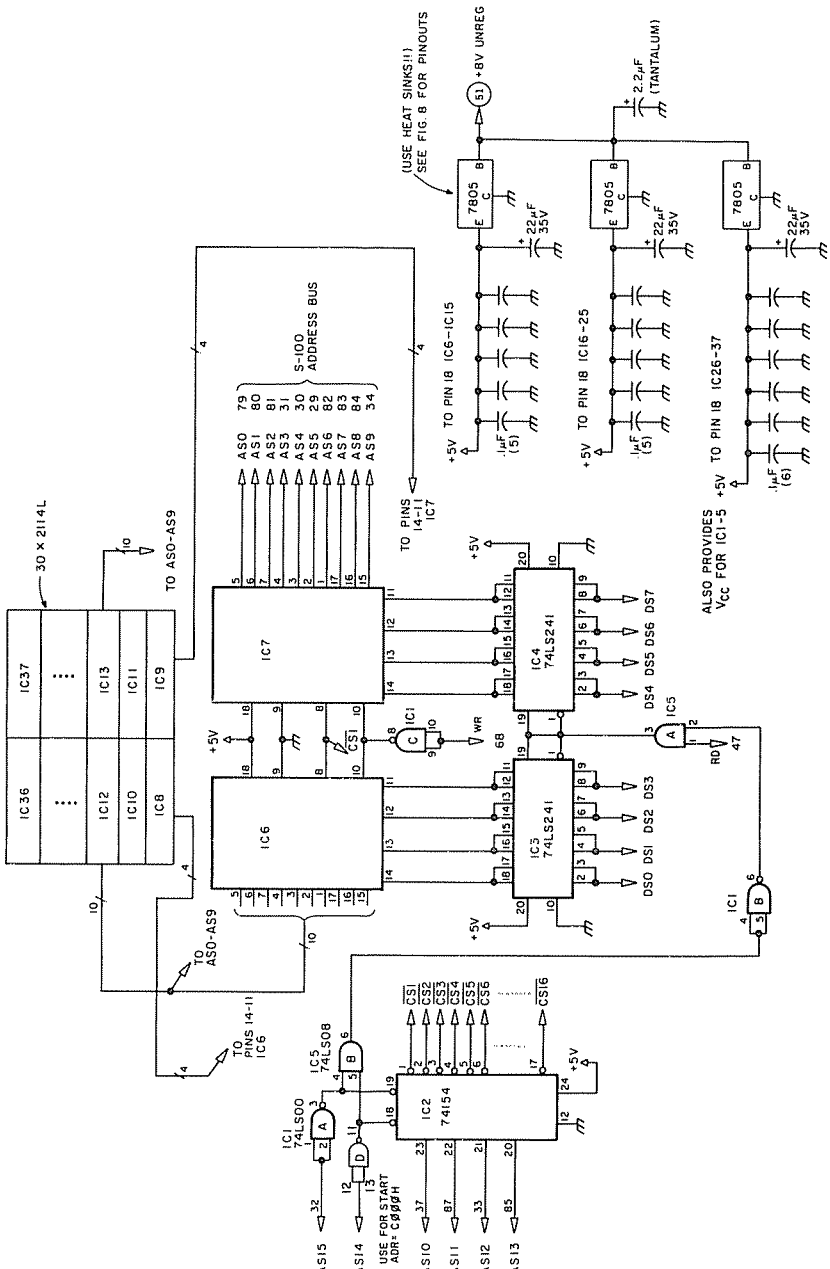


Figure 6. *Static RAM board*

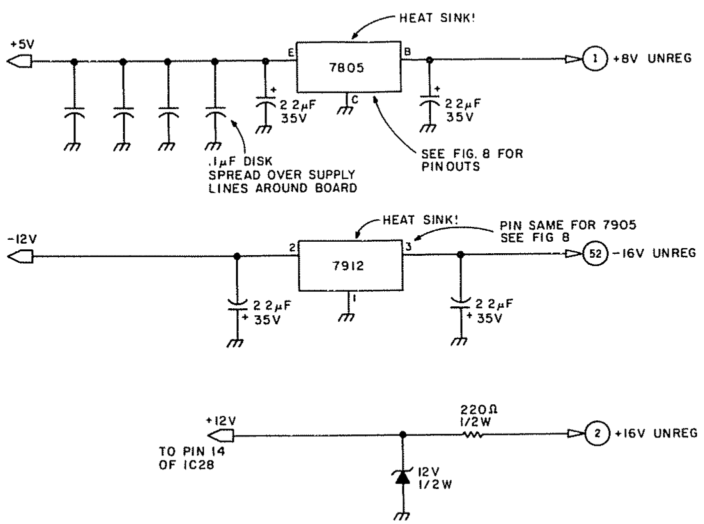


Figure 7. *Power supply—IPC board*

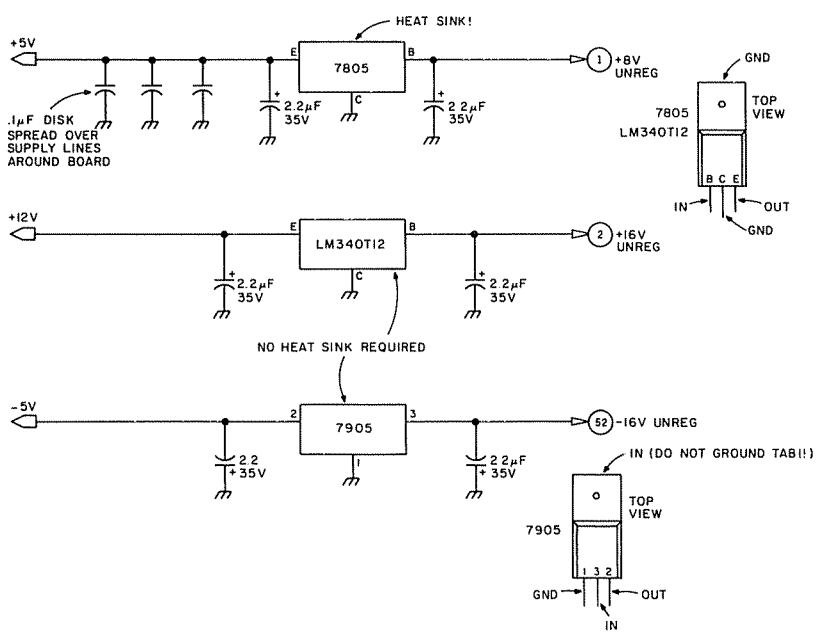


Figure 8. *Power supply—FDC board*

The RAM card is constructed in the same manner as the previous two cards. You should try to locate a .1 uF disc capacitor physically close to every other 2114 and attach it between the +5 V supply (pin 18) and GND (pin 19), keeping the lead lengths as short as possible. The chip select signals may be tested by again writing some small loops in BASIC to access memory within each 1K segment. If you have the disk running when you're testing the memory, this is very simple, since you can now specify a hex address in the BASIC PEEK or POKE statement (i.e., POKE and H8000). When you think you have the memory working properly, you should run the TEST1A/CMD diagnostic on the TRSDOS system disk to verify this. It will tell you at what bit(s) and at what address there is a problem, and it will also specify the problem chip in the Radio Shack expansion interface. If you get the technical manual for the expansion interface, you can cross-reference this information to the S-100 board.

You must have the S-100 mainframe powered on before the MEMORY SIZE? question is answered in order for the TRS-80 to recognize the extra memory available to it. To determine the amount of memory it does have, it writes data to the RAM area and reads it back. If it sees even one wrong bit, it assumes that address is the end of available memory. This can also be used as a diagnostic tool in getting the memory running.

While this expansion interface does not represent the cheapest method for expanding your Model I, it does offer flexibility over the Radio Shack expansion interface and can be built for about the same cost or less. You are also open to the world of S-100 boards available from dozens of sources. Adding

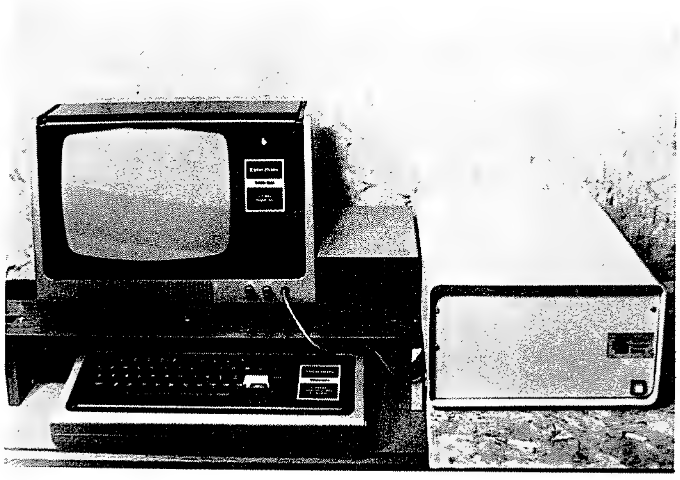


Photo 1. *TRS-80 with expansion box and disk drive*

your own custom hardware is a simple plug-in operation. You don't have to worry about power supplies or special stacking ribbon cable connectors to get everything to plug into the back of the keyboard. The keyboard is isolated from the S-100 bus, so playing on the S-100 bus should never harm the TRS-80. If this is what you have been looking for in a TRS-80 expansion system, then give the Deluxe Expansion Interface a try.

References

(1) Hoeppner, John, "Interface a Floppy-Disk Drive to an 8080A Based Computer." May 1980, *Byte*, pp. 72–102.
(2) Lancaster, Don. *TV Typewriter Cookbook*. Howard W. Sams & Co., Inc., 4300 W. 62nd St., Indianapolis, IN 46268, 1976.
(3) *Intel Data Catalog*. Intel Corporation, 3065 Bowers Ave., Santa Clara, CA 95051, 1980.
(4) *TRS-80 Microcomputer Technical Reference Handbook*. Radio Shack Division of Tandy Corporation, One Tandy Center, Ft. Worth, TX 76102, 1978.
(5) *TRS-80 Expansion Interface Handbook*. Radio Shack Division of Tandy Corporation, One Tandy Center, Ft. Worth, TX 76102, 1980.

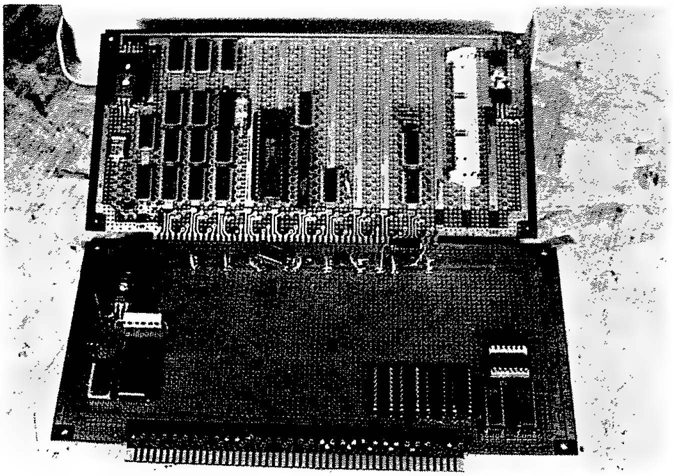


Photo 2. *FDC board (top) minus the external data separator. 16K RAM board (bottom) with the first 2K RAM wired in.*

PORT	FUNCTION IN	OUT
E8	Read modem status	Reset UART
E9	N/A	Set baud rate
EA	Read UART status	Set UART options
EB	Read UART data	Load UART data

Table 1. *RS-232C port functions*

DESIGNATION	PART NUMBER	+5	+12	−12	GROUND
IC1-4	74LS241	20			10
IC5	7408	14			7
IC6	7432	14			7
IC7	7400	14			7
IC8	7432	14			7
IC9	7420	14			7
IC10	Not used				
IC11	74154	24			12
IC12	7420	14			7
IC13	7400	14			7
IC14	COM5016	2	9		11
IC15	AY-5-1013	1		2	3
IC16	7475	5			12
IC17	1489	14			7
IC18	7404	14			7
IC19	7408	14			7
IC20	8212	24			12
IC21	74125	14			7
IC22	74123	16			8
IC23	7420	14			7
IC24	Not used				
IC25	7400	14			7
IC26	7442	16			8
IC27	74125	14			7
IC28	1488		14	1	7
IC29	Not used				
IC30	7432	14			7
IC31	7402	14			7
IC32	1489	14			7
IC33	7432	14			7

Table 2. *IC list for the IPC board*

MEMORY LOCATION	FUNCTION IN	FUNCTION OUT
37EC	Read FDC status	Write FDC command
37ED	Read track reg	Write track reg
37EE	Read sector reg	Write sector reg
37EF	Read disk data	Write disk data

Table 3. *Floppy disk control port functions*

DESIGNATION	PART NUMBER	+5	+12	−5	GROUND
IC1	MC4024	14,13			7,5,9,12
IC2	7474	14			7
IC3–7	7490	5			10
IC8	7474	14			7
IC9	7432	14			7
IC10	7400	14			7
IC11	7474	14			7
IC12	74125	14			7
IC13	74123	16			8
IC14	7407	14			7
IC15	7474	14			7
IC16	INS 1771	21	40	1	20
IC17, 18	8226	16			8
IC19	7408	14			7
IC20	7474	14			7
IC21	7404	14			7
IC22	7420	14			7
IC23	7407	14			7

Table 4. *IC list for the FDC board*

DESIGNATION	PART NUMBER	+5	GROUND
IC1	74123	16	8
IC2	7400	14	7
IC3	7404	14	7
IC4	7400	14	7
IC5	74123	16	8
IC6	7493	5	10

Table 5. *IC list for FDC data separator*

DESIGNATION	PART NUMBER	+5	GROUND
IC1	74LS00	14	7
IC2	74154	24	12
IC3, 4	74LS241	20	10
IC5	74LS08	14	7
IC6-37	2114L	18	9

Table 6. *IC list for the 16K RAM board*

PIN NUMBER	FUNCTION
1	+ 8 volts unregulated dc
2	+ 16 volts unregulated dc
3	WAIT* input to Z-80
4–11	Vectored interrupts—not used by TRS-80
12	External ready #2—not used by TRS-80
13	37E0 device select pulse—interrupt latch
14	37E1 device select pulse—disk select
15	37EC device select pulse—floppy control
16–17	Not used
18	Status disable—not used by TRS-80
19	CMD/CNTRL disable—not used by TRS-80
20	Unprotect—not used by TRS-80
21	Single step—not used by TRS-80
22	Address disable—not used by TRS-80
23	Data out disable—not used by TRS-80
24	Phase 2 clock—not used by TRS-80
25	Phase 1 clock—not used by TRS-80
26	Hold acknowledge—not used by TRS-80
27	Wait—not used by TRS-80
28	Interrupt enable (output)—not used by TRS-80
29	Address 5—AS5
30	Address 4—AS4
31	Address 3—AS3
32	Address 15—AS15
33	Address 12—AS12
34	Address 9—AS9
35/94	Data 1—DS1
36/95	Data 0—DS0
37	Address 10—AS10
38/91	Data 4—DS4
39/92	Data 5—DS5
40/93	Data 6—DS6
41/88	Data 2—DS2

42/89	Data 3—DS3
43/90	Data 7—DS7
44	Machine cycle 1—not used by TRS-80
45	OUT—output device strobe (active high)
46	IN—input device strobe (active high)
47	RD—memory read strobe (active high)
48	Halt acknowledge—not used by TRS-80
49	CLOCK*—inverted phase 2—not used by TRS-80
50	Ground
51	+8 volts unregulated dc
52	−16 volts unregulated dc
53	Sense switch input—not used by TRS-80
54	External clear—not used by TRS-80
55	Real-time clock—not used by TRS-80
56	Status strobe—not used by TRS-80
57	Data input gate #1—not used by TRS-80
58	Front panel ready—not used by TRS-80
59-64	Not used
65	RAS*—row address strobe for dynamic RAM
66	CAS*—column address strobe for dynamic RAM
67	Phantom disable—not used by TRS-80
68	WR—memory write strobe (active high)
69	PS*—project status—not used by TRS-80
70	PROT—protect—not used by TRS-80
71	RUN—not used by TRS-80
72	PRDY—processor ready—used for wait states
73	PINT*—interrupt request
74	PHOLD*—connects to the TEST* TRS-80 signal to force processor busses to a high-impedance state
75	PRESET*—reset line
76	PSYNC—indicates beginning of machine cycle—not used by TRS-80
77	PWR*—used for I/O write operation
78	PDBIN—not used by TRS-80
79	Address 0—AS0
80	Address 1—AS1
81	Address 2—AS2
82	Address 6—AS6
83	Address 7—AS7
84	Address 8—AS8
85	Address 13—AS13
86	Address 14—AS14
87	Address 11—AS11
88-95	See 35-43
96	INTA—interrupt acknowledge
97	SWO*—not used by TRS-80
98	SSTACK—not used by TRS-80
99	POC*—power on clear—not used by TRS-80
100	Ground

Table 7. *S-100 pinout listing for the Deluxe Expansion Interface for the TRS-80*

Interfacing the TRS-80 to the Heath H14 Printer

by George A. Knaust

It didn't take long after purchasing a TRS-80 16K Level II system to realize that a line printer is not only a desirable peripheral, but a necessity when producing software for your system. After looking at specifications, prices, and the fact that Heath was offering a 10 percent discount at the time, I came to the conclusion that the Heath H14 printer was the way to go.

Three months after I placed the order, the kit finally arrived and after several more months of finding the time to work on it, I finally completed it. The Heath documentation was excellent as usual, almost totally without errors.

One of the factors that also influenced my decision to purchase the Heath H14 printer were the ads by Small System Software (SSS) appearing in *Kilobaud Microcomputing* and *80 Microcomputing* for their TRS232 Printer Interface. From the description of this product, there was no reason to believe that this wasn't the way to interface the TRS-80 (without the expansion interface) to the Heath H14 printer.

The only problem after everything was connected, was in implementing line feed. Initially I chose the Auto Line Feed option via a DIP switch on the H14 PC board and answered N to the question: ADD LF AFTER CR(Y/N)? displayed by the SSS program. This did not work, so without further investigation I chose the alternate option, completed the final entry requested by the program, and the printer responded with the correct printout to the LLIST command.

I immediately started making printouts of some of the more important programs I have on cassettes. My interest in mundane things wanes fast, so I started looking into the control commands listed in the *Heath H14 Operator's Manual*, i.e., control of character width and the number of lines per inch. I decided the best way of implementing the control commands was through the concatenation of string values. This would supply the continuous stream of serial data required for the H14 to recognize these commands. The H14 control panel contains a push-button switch labeled WIDE. When this switch is in the depressed position, the printer produces 80 characters per line (10 per inch). When this switch is in the out position, the printer produces 132 characters per line (16.5 per inch).

Table 1, taken from the *Heath H14 Operator's Manual*, shows the control commands required to obtain 80, 96, or 132 characters per line with the WIDE switch either IN or OUT.

WIDTH CODE	SWITCH POSITION WIDE (char/line)	NARROW (char/line)	WIDTH CODE DECIMAL
ESC u CTL-A	80	80	27 117 1
ESC u CTL-D	80	96	27 117 4
ESC u CTL-H	80	132	27 117 8
ESC u CTL-P	96	80	27 117 16
ESC u CTL-T	96	96	27 117 20
ESC u CTL-X	96	132	27 117 24
ESC u SPACE	132	80	27 117 32
ESC u $	132	96	27 117 36
ESC u (	132	132	27 117 40

Table 1.

Also, two choices of line spacing are available on the H14: six lines per inch (activated on power up) and eight lines per inch. Both are software selectable, making it possible to change back and forth under program control. The command codes are as follows:

ESC x for six lines per inch
ESC y for eight lines per inch

The Program Listing is a short program in BASIC that was written to format and send the required commands to the H14. CHR$(27) generates the ASCII code for ESC, CHR$ (117) generates the ASCII code for lowercase u, and A is entered for the desired characters per line from column three under the decimal column in Table 1. B is entered for the number of lines per inch, 120 or 121. CHR$(120) generates the ASCII code for lowercase x, and CHR$(121) generates the ASCII code for lowercase y. This is a straightforward approach to obtaining the desired results. However, initially the printer would not respond with a change in character width or line spacing.

A call to Heath's service center to determine if something was being overlooked or if there could be a bug in the H14 produced no answers. A closer look at the assembler listing of the program POKEd into the high RAM location by the SSS BASIC software revealed that the software tested the code entered, and if it was a command code, it rejected it and looked for the next character.

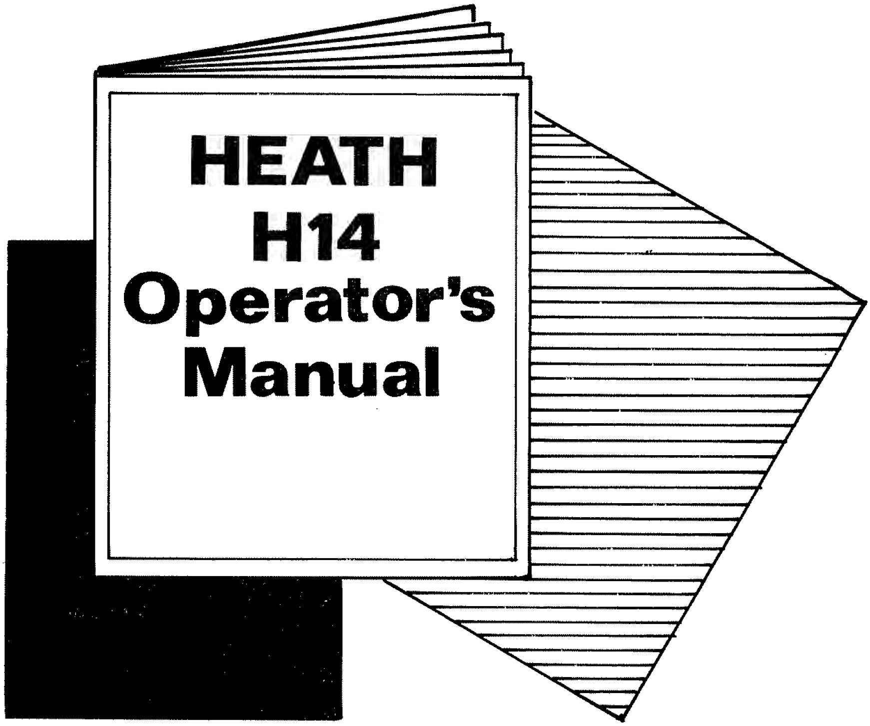
The lines in the assembler listing which perform this test are as follows:

```
CP 20H   ; CONTROL?
RET C    ; YES, RETURN (for next char.)
```

The machine codes for these mnemonics are contained in decimal form in the DATA statements at the end of the SSS BASIC program. Line 1920 contains the decimal code for RET C as 216. Listing this line on the TRS-80

monitor shows the location in the line. By using the EDIT feature of Level II BASIC, the 216 can be deleted and a zero inserted in its place. After this minor change, the SSS program was run. Then the program in the the Program Listing was entered and run, and various decimal codes for character width and line spacing were tried. The H14 printer now responded according to the commands entered. So far, I have not found any harmful effects from this minor change.

In conclusion, I would say the Heath H14 printer interfaced to the TRS-80 via SSS's TRS232 interface is highly recommended for those who would like to add printer capability to their TRS-80. And, in addition, for those so inclined, there is the pride of building the printer yourself and then seeing it work to your satisfaction.



Program Listing 1

```
 10 :
    ' SOFTWARE SELECT OF CHARACTER WIDTH & LINE SPACING
 20 :
    ' FOR HEATH H14 PRINTER BY GEORGE A. KNAUST 5/14/80
 25 CLS
 40 PRINT TAB(10)"SWITCH POSITION"; TAB(36);"WIDTH CODE"
 50 PRINT TAB(8)"WIDE"; TAB(21);"NARROW"; TAB(36);"DECIMAL"
 60 PRINT TAB(9)"80"; TAB(23);"80"; TAB(40);"1"
 70 PRINT TAB(9)"80"; TAB(23);"96"; TAB(40);"4"
 80 PRINT TAB(9)"80"; TAB(22);"132"; TAB(40);"8"
 90 PRINT TAB(9)"96"; TAB(23);"80"; TAB(39);"16"
100 PRINT TAB(9)"96"; TAB(23);"96"; TAB(39);"20"
110 PRINT TAB(9)"96"; TAB(22);"132"; TAB(39);"24"
120 PRINT TAB(8)"132"; TAB(23);"80"; TAB(39);"32"
130 PRINT TAB(8)"132"; TAB(23);"96"; TAB(39);"36"
140 PRINT TAB(8)"132"; TAB(22);"132"; TAB(39);"40"
150 INPUT "ENTER WIDTH CODE FROM TABLE";A
160 INPUT "ENTER 120 FOR 6 LINES/IN. OR 121 FOR 8 LINES/IN.";B
170 A$ = CHR$(27):
    B$ = CHR$(117):
    C$ = CHR$(A):
    D$ = CHR$(B)
180 WC$ = A$ + B$ + C$:
    LPRINT WC$
190 LI$ = A$ + D$:
    LPRINT LI$
200 END
```

TUTORIAL

Saving Machine Language Routines *Below* BASIC

CISAB—Backwards BASIC

Into the 80s
Part VI
Part VII

Saving Machine-Language Routines *Below* BASIC

by Edward B. Beach

Anyone who has programs in BASIC that need machine-language help knows the considerable frustration of having to reserve memory in the TRS-80. I never can remember what MEMORY SIZE to set for the various programs that I run. In addition, some of my programs require loading more than one SYSTEM tape, and then I have to figure out a new protection address for the combined programs. A further complication arises when one expands memory from 16K to 32K, 32K to 48K, or even 4K to 16K. Unless the machine-language routines are fully relocatable, you'll have to move them all to high memory and change all the absolute addresses within the routine. This isn't too much of a problem for people who have an assembler and the source code for the routine. However, I did not have all these nice things when I started out and still find it a nuisance to have to reassemble even short routines.

I have seen quite a few articles on how to move Radio Shack's TBUG machine-language monitor to high memory so programmers could have TBUG resident along with BASIC programs. Even with all of TBUG's shortcomings, it is still a nice, compact (1.5K) utility. One of TBUG's neatest features is that it does not use any of the ROM routines or reserved Device Control Blocks (DCBs), so you're free to do whatever you want with memory without having to worry about disturbing anything in the process. Using the techniques described in this article, you can have TBUG resident along with BASIC programs *without* having to bother about relocating TBUG. It stays right where it was written to reside, at 4380H to 4980H, and has its own stack and workspace.

The way to do this is to reserve low memory for machine-language routines and move the BASIC workspace (programs, variables, and stack) *above* the reserved low memory. Using this technique, we can write machine code using absolute addresses in any size memory, since the machine-language code will always be at the same place in memory—regardless of the amount of memory in the system. The one problem is that low memory starts at different locations for different TRS-80 models and operating systems.

Figure 1 shows why this is so. In the Model I, BASIC workspace begins at 42E9H, while in the Model III it starts one page (256 bytes) higher, at 43E9H. Disk BASIC uses a starting location somewhere in page 6BH. In all models and systems, however, there are two very important pointers that tell the operating system just where the BASIC workspace is located. We will use these two pointers to move the workspace to allow us room at the

low end of memory for our own purposes. Just remember that routines written in machine language for the Model I will not necessarily run in a Model III unless absolute addresses within the routine are adjusted upward by 256 bytes. Disk BASIC is also an entirely different matter.

Of course there are other pointers used by BASIC that are also important. However, we do not need to concern ourselves with these pointers, because they are automatically adjusted any time a BASIC program is run or CLOADed. The command NEW will also reset all of the various pointers, provided the two important pointers mentioned before are correctly set.

The two pointers we must manipulate are at 40FFH and 40A4H. The pointer at 40FFH points to the first entry in the BASIC workspace. On power-up this is initialized to an address of 42E8H in the Model I and 43E8H in the Model III. The pointer at 40A4H is initialized to an address one byte past this address: 42E9H and 43E9H in the Model I and Model III respectively. In addition, the actual data at the location pointed to by the vector at 40FFH must be zero. If this location contains a non-zero value, you will get an error message for any command you type in.

To simulate an empty BASIC workspace, the location pointed to by the vector at 40A4H (and the following location as well) must also be set to zero. Otherwise the two bytes starting at this location are assumed to be a link address to the next line of a BASIC program. Unless these locations contain a legitimate link address, you can get some peculiar results if you run or list a program.

As a simple example of how we might reserve low memory, let's say we would like the beginning of the BASIC workspace to be relocated to 4400H. This would leave locations from 42E9H to 43FFH free for us to use and protected from BASIC. (This example, and most of those which follow, will assume that we are discussing the Model I. Adjust everything upward by 256 if you are dealing with the Model III.) We would have to make the address (pointer) at 40FFH be 4400H, and the address (pointer) at 40A4H be 4401H. In addition, we would have to be sure that the data at locations 4400H, 4401H, and 4402H is zero. It's that simple.

One of the most straightforward ways to enter a machine-language program into a TRS-80 is to have a BASIC program READ the data bytes from DATA statements and POKE them into memory. I have used this technique many times and have always been appalled by the fact that the final machine-language program occupied far less memory than data elements needed to produce the program. The data is stored by BASIC in DATA statements as ASCII characters. Each machine-language byte is encoded as from one to three ASCII characters (0 to 255) in the DATA statement. In addition, the commas separating the numbers in the DATA statements each take up another byte of memory.

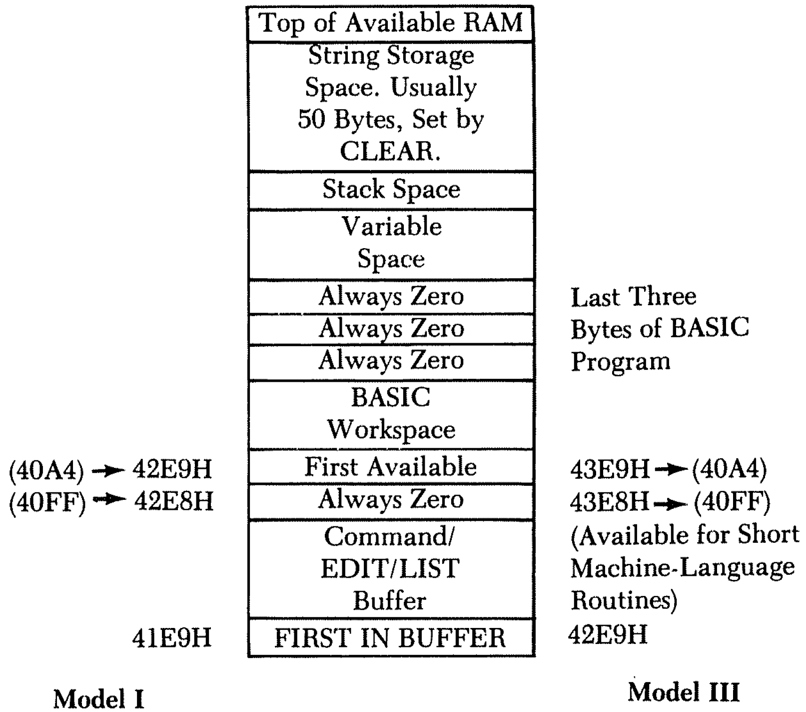


Figure 1. *BASIC memory map of Model I and Model III*

It always seemed to me that a neat way to use BASIC to POKE machine-language data into memory would be to overlay the DATA statements which produced the code with the resulting machine code. This is fairly easy to do if we put the DATA statements that make up the machine-language program at the *beginning* of the BASIC program and then use a FOR-NEXT loop to read the data and POKE it into the locations just read. After the machine code is POKEd into memory, we set up the two pointers mentioned before, initialize the first three bytes of the new BASIC workspace to zero, and then execute a NEW command. This will wipe out the BASIC program, which is no longer needed, and reset all the necessary memory pointers.

Using this approach, we do not reserve any more of low memory than is actually required for the machine-language code. In addition, if we need to enter more machine-language code, we can use exactly the same technique again and reserve additional low memory starting in the relocated BASIC workspace. We actually sacrifice five bytes of memory every time we do this since we will have to preserve these bytes while the BASIC program reads and POKEs. The five bytes constitute the link address to the next line number (first two bytes), the line number of the DATA statement (next two

bytes), and the DATA keyword (one byte). With the exception of these five "wasted" bytes, the READ and POKE technique is *very* conservative of memory.

Program Listing 1 is an example of using this technique of POKEing machine code over DATA statements. Lines 10 and 20 are the DATA statements that hold the machine code. The machine-language program represented by the two DATA statements is a simple driver for the TRS-80 video to allow the display of lowercase characters. It is a fully relocatable routine and occupies 30 bytes of memory.

If you count the number of elements in the two DATA statements you will see that there are 31. The extra element tells the POKEing routine how many elements to read and POKE. This is the first number in line 10. Line 30 sets all variables used to integer to speed things up. Line 40 picks up the two bytes at 40A4H and 40A5H (16548 and 16549) to find the beginning of the current BASIC workspace. These two bytes are assigned to AL and AH (Address Low and Address High) after adding five to the low-order address byte. The subroutine at line 510 checks AL for overflow past 255 and adjusts AH and AL if needed.

The address now contained in AH and AL is the location where we will begin POKEing the machine-language program. This same address must be substituted for the address in the video device control block (DCB) at 16414 and 16415. Line 50 takes care of this for us. Line 60 reads the byte count into N for the limit of the FOR-NEXT loop in lines 70 through 90. Line 80 reads the data, POKEs it into memory and then increments AL (and AH if necessary). Subroutines at lines 600 and 500 handle these operations.

After all the data has been POKEd into memory, AL and AH point to the byte just following the last byte POKEd in. All that remains is to fill this location and the next two locations with zeros and reset the two BASIC pointers to their new values. Lines 90 through 120 handle this task. Notice that the last instruction in line 120 is NEW, which will wipe out the BASIC program and restart BASIC with a clean slate. The entire BASIC program from line 30 to the end (with the exception of line 50) can be used as a general-purpose routine for POKEing machine-language routines into low memory. For linking BASIC programs to machine-language programs, line 50 should POKE AL and AH into locations 16526 and 16527, respectively. These are the USR address pointer locations.

Figure 2 shows how the machine-language program of Program Listing 1 is actually POKEd into memory to overlay the data elements of line 10. The first data byte (221) goes into the space between the DATA keyword and the 3 of the 30 in the first data element. The next data byte (110) replaces the 3 of 30, and the next data byte (3) replaces the 0 of 30, and so on. The last data byte replaces the 5 of the 154 data element. The next three locations (4,4) are

the three nulls used to mark the BASIC workspace with no program present. The location of the middle null (the comma between 154 and 4) is the new beginning of BASIC workspace. Notice how little space this routine occupies compared to the BASIC program and DATA statements!

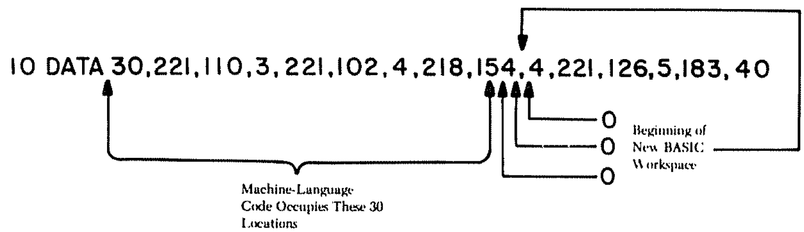


Figure 2. *Machine-language code overlays data elements.*

For very long machine-language programs, it is conceivable that the machine code will eventually be POKEd into locations in the BASIC program that could possibly cause trouble, such as at the end-of-line token (a null), the link address or line number of the next DATA statement, or the next DATA keyword. In practice this is never going to create a problem. The reason for this is that BASIC maintains a separate DATA pointer. As long as all the READs are done at one time (as they will be here), the DATA pointer will be stepped forward quite regularly. It will have reached the end of the first DATA statement *long* before the POKEd data reaches this physical location. It will find the next DATA statement and be quite happy again for a long time. It is perfectly all right to go ahead and destroy (write over) the preceding end-of-line marker, line link address, line number, and keyword. The DATA pointer always moves on!

Another Hiding Place in Low Memory

For very short machine-language routines like the one in Program Listing 1, there is an even better way to store programs in low memory. If you look again at the memory map in Figure 1, you will see that there is an area of memory that is 256 bytes long reserved for the command/LIST/EDIT buffer. This buffer is used to hold BASIC command inputs and lines of BASIC programs when they are edited or listed. In most instances, very little of this buffer is ever used. If you were to have an extremely long line in BASIC, it could fill up the buffer. Most of us never use anywhere near all of the buffer. In fact, I have never written a program myself (or listed a commercial BASIC program) that used more than half of this buffer. This means that

perhaps 100 or so bytes at the upper part of the buffer are unused. There is most certainly room for the 30-byte video driver program.

Using the command buffer for machine-language program storage has a nice feature associated with it: If you should ever have to totally reset the computer (without turning off power), your machine-language program will still be there. This is not necessarily true of SYSTEM programs in reserved high memory or for programs POKEd into low memory as described earlier. A return to MEMORY SIZE? will reset everything to power-on conditions. But programs in the command buffer will remain. All you need to do is reset the pointers (usually USR at 16526, 16527) to recover the machine-language program.

Program Listing 2 shows how the video driver can be POKEd into this hiding place. Notice that this program assigns the starting address (17096 or 42C8H) absolutely, without checking BASIC pointers. In addition, the program is not automatically erased with a NEW command since the DATA statements have not been written over. Instead, the last part of the program reminds the user that he or she can recover the video driver routine by POKEing appropriate values into two locations. The entire BASIC workspace can now be used by typing NEW to remove the BASIC program. Keep this technique in mind if you have short machine-language routines to keep on tap. You might even fit in two or three USR routines, changing the USR pointer as required from the calling BASIC program.

Hiding TBUG in Low Memory

If you use TBUG, here is how you can hide it below the BASIC workspace. Use TBUG to enter the short program shown in Program Listing 3. Then use TBUG's P command to make a SYSTEM tape with a starting address of 436EH, going to 4980H, with a default entry address of 436EH. You can use any program name you like for the program. I chose TBUGL for originality.

When you load the program using the SYSTEM command, start the program by typing / ENTER in response to the SYSTEM prompt. You will then get the BASIC READY prompt and you're off and running. Even in a 4K system you will have almost 1.5K of memory left for BASIC programs above TBUG. In addition, the space from 42E9H to 4380H is available to you, free of charge. You can enter TBUG from BASIC at any time by using the SYSTEM command followed by /17312 ENTER, just as always.

If you would rather have your modified TBUG begin execution in TBUG rather than in BASIC, change the last two bytes of Program Listing 3 to A0 43. To get from TBUG to BASIC, use TBUG's J command to jump to address 1A19H. This will get you the READY prompt from BASIC. Be sure to type

NEW, CLOAD, or RUN to reset all the necessary pointers. Otherwise you will find that BASIC thinks it doesn't have a great deal of memory; TBUG has moved the stack to 4980H, and BASIC won't like this! NEW, CLOAD, or RUN (even with no BASIC program present) will straighten things out.

You can use the 18-byte program in Program Listing 3 all by itself to set aside low memory for machine-language programs. Change the third and fourth bytes to any address you would like to establish as the beginning of BASIC workspace, remembering that the least significant half of the address comes first. Now, everything from 42E9H to the address you select will be protected from BASIC. It's simple, easy to do, and easy to tack on to any machine-language routine you want to use with BASIC, or all by itself.

Program Listing 1

```
10 DATA 30,221,110,3,221,102,4,218,154,4,221,126,5,183,40
20 DATA 1,119,121,254,32,218,6,5,254,128,210,166,4,195,125,4
30 DEFINTA-Z
40 AH = PEEK(16549):AL = PEEK(16548) + 5:GOSUB510
50 POKE16414,AL:POKE16415,AH
60 READ N
70 FOR I = 1TON
80 READ D: GOSUB600:GOSUB500
90 NEXTI:D = 0
100 GOSUB600:POKE16639,AL:POKE16640,AH:GOSUB500
110 GOSUB600:POKE16548,AL:POKE16549,AH:GOSUB500
120 GOSUB600:NEW
500 AL = AL + 1
510 If AL>255 THENAL = AL - 256:AH = AH + 1
520 RETURN
600 POKE 256*AH + AL,D:RETURN
```

Program Listing 2

```
10 FORI = 0T029
20 READ D:POKE 17096 + I,D
30 NEXT I
40 CLS:PRINT"TO RESTORE LOWERCASE AFTER SYSTEM RESET, ENTER THE
   FOLLOWING:"
50 PRINT:PRINT"POKE 16414,200:POKE 16415,66"
60 POKE 16414,200:POKE 16415,66
70 DATA 221,110,3,221,102,4,218,154,4,221,126,5,183,40,1
80 DATA 119,121,254,32,218,6,5,254,128,210,166,4,195,125,4
```

Program Listing 3

```
436E: AF 21 80 49 77 22 FF 40 23 77 22 A4 40 23 77 C3 19 1A
```

CISAB: Backwards BASIC

by C. Brian Honess

That's right—backwards BASIC. In this chapter we'll learn a whole new language, modestly named HONESS (take your choice of acronym: Here's Our New, Easy, Super System, or maybe Help Out Noteworthy Enthusiastic Successful Students, or maybe Handy Official Notation Employing Sophisticated Subjects). Whatever you call it, HONESS is a machine language, which is fed into your BASIC interpreter along with the data, at which point BASIC takes over. You write your instructions in machine language. The important thing about HONESS is that you will see how a machine language works, and you will be able to expand the operation codes, as well as expand and experiment to your heart's content. Let's see what it is all about.

HONESS can be run on almost any computer. You could just as easily code a FORTRAN program to accept HONESS instructions and data. We'll assume that the computer running HONESS programs has 99 storage locations. Each of these 99 storage locations can hold eight decimal digits, divided into four groups of two digits each (see Figure 1). We'll call each of the eight-digit groups a word, so we'll need a 99-word machine. Each of these 99 words can store either an instruction or a data value, and the largest data value that our language will handle is 99999999. We'll have to make a few concessions to keep things simple at first, so let's work with just whole numbers for now.

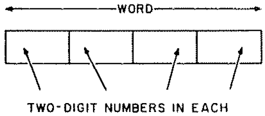


Figure 1

The first two digits of the word contain the op (operation) code, and the other three two-digit parts contain addresses, or operands. Some op-codes are going to need all three of the addresses, and others will require just one, two, or none at all. If a particular address isn't required, we'll just fill it with two zeros. We'll start you off with the five basic arithmetic operations (add, subtract, multiply, divide, and exponentiate), and a read and write command, plus a halt instruction. We'll also need an unconditional branch instruction, and then a high, low, equal compare

technique. This set of operation codes will make a fairly powerful language.

We're going to use base 10 numbers for each of the op-codes, and the addresses and data, just to make things easier for you. Of course, true machine language would be just ones and zeros (binary). If we used true binary, each word would have to be able to hold 28 binary digits, in four groups of seven, because it takes seven binary bits to hold the largest

Op-code	Meaning	Address Portion			Meaning
01	Add	A1	A2	A3	Add contents of address A1 to the contents of address A2 and store the result in A3
02	Subtract	A1	A2	A3	Subtract the contents of A2 from A1 and store the result in address A3
03	Multiply	A1	A2	A3	Multiply contents of A1 by contents of A2 and store result in A3
04	Divide	A1	A2	A3	Divide contents of A1 by contents of A2 and store result in A3
05	Read	A1	00	00	Read into location A1
06	Print	A1	00	00	Print from location A1
07	Exponentiate	A1	A2	A3	Raise contents of A1 to the power stored in A2 and store result in A3
08	Branch	A1	00	00	Unconditional branch to location stored in A1
09	Halt	00	00	00	Stop
10	Equal compare	A1	A2	A3	If contents of A1 = contents of A2, GOTO location in A3
11	Low compare	A1	A2	A3	If contents of A1 < contents of A2, GOTO location in A3
12	High compare	A1	A2	A3	If contents of A1 > contents of A2, GOTO location in A3

Table 1. *Op-codes found in HONESS*

decimal number that is two-digits long (99), so instead of writing 28-bit binary numbers for each word, let's use eight-digit decimal numbers.

Table 1 shows the various op-codes, and which address portions are required for them. We'll be writing all our programs using these op-codes. You can expand on them if you like, and put in all sorts of special-purpose functions.

Let's look at a couple of instructions now, break them into the four two-digit groups, and see what they mean.

04586312

04 58 63 12

The op-code is 04, which means divide. The three addresses say: Divide the number stored in location 58, by the number stored in location 63, and store the answer at location 12.

12171893

12 17 18 93

This is a high compare op-code, so if the contents stored at location 17 are greater than the contents stored at location 18, then we go to the instruction number stored in location 93.

Simple Programs

We're ready to write a simple program now, but there is one little problem remaining—how to load the program into the desired storage locations. We might not want to store the program in the first n storage locations, and we might want to store constants, etc. in high storage without having to go to the trouble of storing zeros in all the unused locations between the end of the program and the high-storage area. To get around this, we're going to attach a two-digit loader in front of each of the eight-digit words. Therefore, if we want to store the instruction 08151617 into location 73, it would become: 7308151617.

For our first program, let's consider the problem of reading in two numbers, adding them together, and then printing out the result. Here is a HONESS program to do this. I've coded it on the left, the way it was written and broken it down into the component parts on the right, so we can see them more easily to talk about them.

0105720000	01	05	72	00	00
0205730000	02	05	73	00	00
0301727358	03	01	72	73	58
0406580000	04	06	58	00	00
0509000000	05	09	00	00	00

The first two-digit number drops off after loading, and 05720000 is in storage location 1, 05730000 is in 2, and so on. After loading, the computer is directed to return to storage location 1 and begin executing the program. The 05 op-code says to read a data value that is stored in location 72. The second instruction is also a read, and it reads into location 73. The third instruction says to add the contents of 72 to the contents of 73 and store the result in 58. The fourth instruction says to print out the contents of location 58, and the last instruction is a STOP instruction. There

is nothing sacred about the storage locations, 72, 73, and 58. They were just unused locations in our 99-word memory. We could have used any other numbers between six and 99. We couldn't use one through five, because those locations hold the program. Therefore, they can't be used for data values.

Let's consider the problem of finding the average of n sets of three numbers each. In other words, we have a group of students (any number between one and the largest number your machine can calculate in an evening), and each student has three exam grades. You want to compute the average of each set of three exam grades, and then return to consider another student, and so on. Let's draw a flowchart for this infinite loop program (Figure 2):

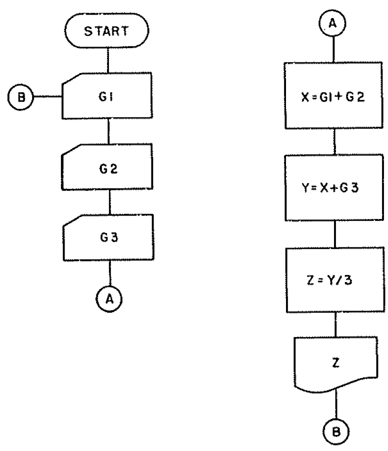


Figure 2

Now, I'll code the HONESS program and write what each line does to the right of it. Also, I'll break down each instruction into components, but of course, when keyed into the machine, these would all look like 10-digit numbers.

Loader	Op-code	A1	A2	A3	Comments
01	05	23	00	00	Read 1st number into 23
02	05	24	00	00	Read 2nd number into 24
03	05	25	00	00	Read 3rd number into 25
04	01	23	24	26	Add 1st and 2nd numbers and store in 26
05	01	25	26	26	Add 3rd number to total in location 26
06	04	26	99	26	Divide total in 26 by the constant in 99, and put answer back in 26

07	06	26	00	00	Print contents of 26
08	08	01	00	00	Go to 1st instruction
99	00	00	00	03	Constant used for division

Next, let's consider the equation $y = ax^2 + bx + c$, by reading in some values for a, b, c, and x, and calculating and printing y. I won't bother with a flowchart, because you should be able to figure this one out. Notice that the arrows in the comments mean "put into location number," and that I find x^2 by multiplying x by itself.

Loader	Op-code	A1	A2	A3	Comments
01	05	21	00	00	Read a → 21
02	05	22	00	00	Read b → 22
03	05	23	00	00	Read c → 23
04	05	24	00	00	Read x → 24
05	03	24	24	25	x^2 → 25
06	03	21	25	26	ax^2 → 26
07	03	22	24	27	bx → 27
08	01	26	27	28	$ax^2 + bx$ → 28
09	01	23	28	29	$ax^2 + bx + c$ → 29
10	06	29	00	00	Print ← 29
11	09	00	00	00	Stop

An Expanded Averaging Program

Now, let's expand our average-of-three-numbers program, to find the average of any number of numbers. We'll have to use the trailer principle to get out of the reading loop, so we'll choose a trailer value of 99999999, the largest number we can hold in our storage word (therefore, we can't have the number 99999999 as one of the numbers we're finding the average of).

We'd better flowchart this one, so it'll be easier to follow (see Figure 3).

Here's the program in HONESS, written from the flowchart shown in Figure 3. Compare each instruction in the code with the appropriate flowchart block, to gain an understanding of how this trailer-triggered, loop-with-an-exit program works.

0105320000	Read x → 32
0210329606	If x = 99999999 GOTO 06
0301979898	Add 1 to C (counter)
0401329999	Add x to S (sum)
0508010000	GOTO 01
0604999851	S / C → 51
0706510000	Print ← 51
0809000000	Halt
9700000001	Constant for incrementing counter, C
9800000000	Counter, C, for number of values read
9900000000	Store sum of x values here

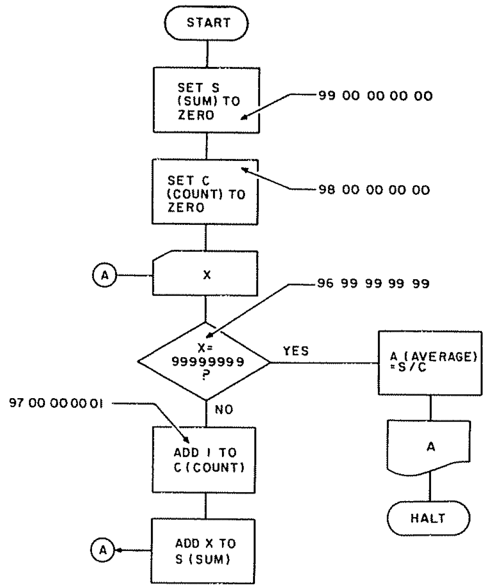


Figure 3

Finding the Largest Number

For our last example, let's consider something with a few more decision blocks. We'll read in three numbers, and print out the largest one. The three numbers can be all the same, all different, (or anything in between), and in any order.

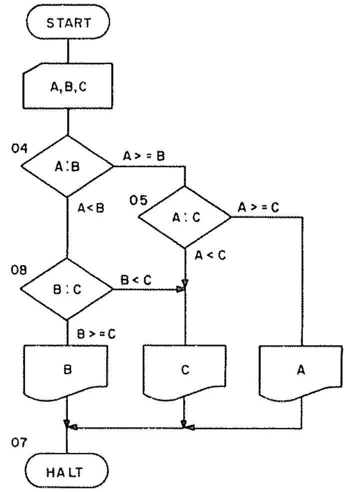


Figure 4

There are two ways to solve this problem, and a flowchart and program are delineated for each. I'll write a few comments to the right of each program, and also separate the component parts, so you can study them more easily. The first solution method is a real mind-boggler (see Figure 4), with decision blocks, GOTOs, and the like, all over the place. The second is a little more organized (see Figure 5), in that we consider the first number to be the largest, no matter what value it has, then we change it to another value if we find one that is larger.

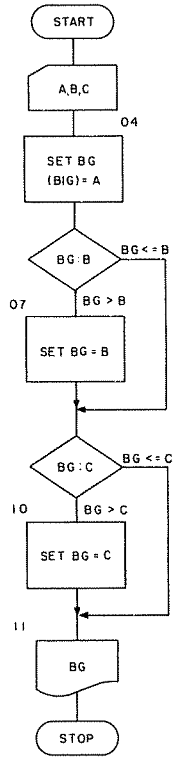


Figure 5

Here is the program in HONESS, written from the flowchart shown in Figure 4:

01	05	61	00	00	Read A → 61
02	05	62	00	00	Read B → 62
03	05	63	00	00	Read C → 63
04	11	61	62	08	If A< B GOTO 08
05	11	61	63	11	If A< C GOTO 11
06	06	61	00	00	Print A
07	09	00	00	00	Halt

08	11	62	63	11	If B<C GOTO 11
09	06	62	00	00	Print B
10	08	07	00	00	GOTO 07
11	06	63	00	00	Print C
12	08	07	00	00	GOTO 07

Here is the HONESS program written from the flowchart in Figure 5:

01	05	61	00	00	Read A → 61
02	05	62	00	00	Read B → 62
03	05	63	00	00	Read C → 63
04	01	61	98	99	Move A to BG (this is done by adding zero)
05	11	99	62	07	If BG<B GOTO 07
06	08	08	00	00	GOTO 08
07	01	62	98	99	Move B to BG
08	11	99	63	10	If BG < C GOTO 10
09	08	11	00	00	GOTO 11
10	01	63	98	99	Move C to BG
11	06	99	00	00	Print ← BG
12	09	00	00	00	Halt
98	00	00	00	00	Constant zero, used for converting an ADD to a MOVE instruction.
99	00	00	00	00	Storage for BG

The HONESS Program

Now we're ready for the actual HONESS program. The listing is fairly long, but I've included lots of REMarks. I guess you don't really have to key them in, but they could be handy to have with the Program Listing, since they contain all of the features of the language, plus the instructions on how to load any programs you write in HONESS. The first 62 lines (10 through 71) are REMarks, and we've already discussed everything through line 63, so let's pick up the discussion with line 64. We need to have some way of telling the machine when the program has been read, so that it won't continue reading DATA statements, thinking they are program instructions, and try to execute them. We'll key a value of zero into the loader to mean we've come to the end of the program instructions, and any other DATA statements containing data, rather than HONESS instructions. Lines 901 through 998 will be used for entering the program and any data values, and REMark 70 shows a typical DATA line. This happens to be an instruction that is being loaded into location number 21. It is an ADD instruction, and says to add the number in 10 to the number in 11 and store the answer in 17.

Line 74 defines the variables M and X as being double precision. This is necessary, since we're dealing with 10-digit numbers going in, and the usual six-place BASIC just won't be enough. Line 75 sets up the 99 storage

locations. Line 76 reads a DATA statement, and loads it into location X. In line 77, it strips off the first two digits, which you'll recall is the loader. This tells it which of the 99 locations to store that particular instruction in. We strip off the first two digits by dividing by 100000000 in line 77. In line 78, we check to see if the loader equals zero, because if it does, then this is the last program instruction, and loading has been completed. If the last instruction hasn't been found, line 79 loads the instruction in the correct storage location, we then go to the read statement again, from line 80, and read the next instruction. This process continues until the zero is found in the loader.

Lines 81 through 88 simply print out a big long table of how all 99 storage positions look after the HONESS program has been loaded. You may want to put some sort of prompt and an IF-THEN statement in here, so that you can skip this part unless you particularly want this dump for debugging purposes. Line 88 lets us know that the execution phase has begun.

Lines 89 and 90 take us back to the first instruction to begin execution. "Why did you need to set N = 0 and then N = N + 1 instead of just saying N = 1?" I hear you ask. Well, if I'd done it that way, I would have had to put an N = N + 1 before every GOTO 90 later in the program. Line 91 pulls out the particular instruction we'll be working on, and loads it into X. This insures that the original instruction remains intact, and we just make a duplicate of it.

Lines 92 through 95 separate the eight-digit number in X into its four component parts (A0, A1, A2, and A3–A0 is the op-code, remember). Then, in line 100, we look at the op-code and go to one of the 12 listed statements, depending upon the value of the op-code. If you add additional op-codes to the program, you'll have to add more line numbers to the ON. . .GOTO, and add the coding for your particular op-code at those line numbers. I think you'll see how easy it is to add op-codes from looking at the 12 I've included. Here is a list of just some of the op-codes you could add easily:

Square root	Reciprocal
Sine	Cube root
Absolute value	Arc sine
Base e log	Arc cosine
Base 2 log	Arc tangent
Base 10 log	Cosine
Tangent	Hyperbolic sine
Hyperbolic cosine	Hyperbolic tangent
Change sign	Move

Depending on the op-code, we GOTO a line number somewhere between 110 and 220, and at each of the locations the operation is performed, and we return to increment N and look at the next instruction.

Eventually we run into the 09 HALT op-code, and that results in a GOTO 190, which then immediately directs us to line 400. At line 400, a dump of all 99 locations occurs, so that you can see the ending values stored in each location. Again, you may want to print this only after a YES answer to a prompt and question. It should be easy for you to add this feature.

Now you're ready to try the example programs on the previous pages, before you write some of your own. I'll code the necessary DATA statements for the first program we considered, wherein we simply read in two numbers, added them, and printed the result.

```
901  DATA  0105720000
902  DATA  0205730000
903  DATA  0301727358
904  DATA  0406580000
905  DATA  0509000000
906  DATA  0000000000     (Line 906 is the statement with
907  DATA  2              the 00 loader that separates
908  DATA  5              the program from the data.)
```

This could all be put on fewer lines by keying in the numbers with commas between them, perhaps five or six per line, but I think this way it makes it easier to debug and visualize. Since my data values for the program are 2 and 5, the answer, 7, should be printed after the program prints EXECUTION HAS BEGUN! and before it does the POST-MORTEM DUMP OF STORAGE.

Program Listing

```
1 REM ******************************************************
2 REM *                                                    *
3 REM *  "CISAB ( THAT'S BACKWARDS BASIC FOLKS! )          *
4 REM *                                                    *
5 REM *   BY:      C. BRIAN HONESS                         *
6 REM *            COLLEGE OF BUSINESS ADMINISTRATION      *
7 REM *            UNIVERSITY OF SOUTH CAROLINA            *
8 REM *            COLUMBIA, SC  29208                     *
9 REM ***                                                ***
10 REM     HONESS IS A MACHINE LANGUAGE CAPABLE OF BEING RUN ON
11 REM  MOST DIGITAL COMPUTERS.  IT HAS A MEMORY CAPACITY OF 99
12 REM  WORDS.  EACH WORD CONTAINS EIGHT DECIMAL DIGITS.  THESE
13 REM  DIGITS ARE DIVIDED INTO FOUR GROUPS OF TWO DIGITS EACH.
14 REM  WE SHALL DENOTE THESE FOUR GROUPS AS A0, A1, A2, AND A3
15 REM  WHERE A0 IS THE OPERATION CODE AND A1, A2, AND A3 ARE
16 REM  USED TO CONTAIN THE LOCATIONS OF THE OPERANDS.  FOR
17 REM  EXAMPLE, LET A0, A1, A2, AND A3 BE 01, 14, 15, AND 20.
18 REM  01 INDICATES THAT THE OPERATION IS ADDITION.  THE
19 REM  CONTENTS OF WORD 14 ARE ADDED TO THE CONTENTS OF WORD
20 REM  15, AND THE RESULT IS STORED IN WORD 20.
21 REM
22 REM     THE FOLLOWING ARE HONESS OPERATION CODES:
23 REM
24 REM  01 - ADD            01 A1 A2 A3  MEANS ADD CONTENTS OF
25 REM                                   A1 TO CONTENTS OF A2
26 REM                                   AND STORE RESULT IN A3
27 REM  02 - SUBTRACT       02 A1 A2 A3  MEANS SUBTRACT CONTENTS
28 REM                                   OF A2 FROM CONTENTS OF
29 REM                                   A1 AND STORE RESULT IN A3
30 REM  03 - MULTIPLY       03 A1 A2 A3  MULTIPLY CONTENTS OF A1
31 REM                                   BY CONTENTS OF A2 AND
32 REM                                   STORE RESULT IN A3
33 REM  04 - DIVIDE         04 A1 A2 A3  DIVIDE CONTENTS OF A1
34 REM                                   BY CONTENTS OF A2 AND
35 REM                                   STORE RESULT IN A3
36 REM  05 - READ           05 A1 00 00  READ INTO LOCATION A1
37 REM  06 - PRINT          06 A1 00 00  PRINT FROM LOCATION A1
38 REM  07 - EXPONENTIATE   07 A1 A2 A3  RAISE CONTENTS OF A1 TO
39 REM                                   THE POWER STORED IN A2
40 REM                                   AND STORE RESULT IN A3
41 REM  08 - BRANCH         08 A1 00 00  UNCONDITIONAL BRANCH TO
42 REM                                   LOCATION STORED IN A1
43 REM  09 - HALT           09 00 00 00  STOP AT THIS LOCATION
44 REM  10 - COMPARE  =     10 A1 A2 A3  IF CONTENTS OF A1  =
45 REM                                   CONTENTS OF A2, GO TO
46 REM                                   LOCATION IN A3
47 REM  11 - COMPARE  <     11 A1 A2 A3  IF CONTENTS OF A1  <
48 REM                                   CONTENTS OF A2, GO TO
49 REM                                   LOCATION IN A3
50 REM  12 - COMPARE  >     12 A1 A2 A3  IF CONTENTS OF A1  >
51 REM                                   CONTENTS OF A2, GO TO
52 REM                                   LOCATION IN A3
53 REM
54 REM     IN ORDER TO READ THE PROGRAM INTO THE COMPUTER, EACH
55 REM  WORD HAS A TWO-DIGIT PREFIX ATTACHED TO IT.  THIS TWO-
56 REM  DIGIT PREFIX IS CALLED A LOADER.  ITS PURPOSE IS TO
57 REM  PLACE EACH WORD IN A LOCATION SPECIFIED BY THE PROGRAMMER.
58 REM  IF THE INSTRUCTION WORD IS TO BE STORED IN LOCATION 21,
59 REM  THE LOADER AND INSTRUCTION WOULD APPEAR AS:
60 REM  21 01 10 11 17      ( THE INSTRUCTION IS, OR COURSE, TO
61 REM  ADD THE CONTENTS OF 10 TO THE CONTENTS OF 11 AND STORE
62 REM  THE RESULT IN 17.  )
63 REM
64 REM     A "00" KEYED INTO THE FIRST TWO COLUMNS OF A DATA
65 REM  LINE, SIGNIFIES THE END OF THE PROGRAM.  DATA, IF ANY,
66 REM  WILL FOLLOW.
67 REM
```

Program continued

```
68 REM      A TYPICAL DATA LINE WOULD THEN LOOK AS FOLLOWS:
69 REM
70 REM           902 DATA 2101101117
71 REM
74 DEFDBL M,X
75 DIM M(99)
76 READ X
77 LD = INT ( X / 100000000 )
78 IF LD = 0
    THEN
     81
79 M(LD) = X - ( LD * 100000000 )
80 GOTO 76
81 PRINT
82 PRINT "MEMORY DUMP AFTER LOADING COMPLETED:"
83 PRINT
84 FOR I = 1 TO 99
85  PRINT I, M(I)
86  NEXT I
87 PRINT
88 PRINT "EXECUTION HAS BEGUN!"
89 N = 0
90 N = N + 1
91 X = M(N)
92 A0 = INT ( X / 1000000 )
93 A1 = INT ( X / 10000 ) - ( A0 * 100 )
94 A2 = INT ( X / 100 ) - ( A1 * 100 ) - ( A0 * 10000 )
95 A3 = X - ( A2 * 100 ) - ( A1 * 10000 ) - ( A0 * 1000000 )
100 ON A0 GOTO 110,120,130,140,150,160,170,180,190,200,210,220
110 M(A3) = M(A1) + M(A2)
111 GOTO 90
120 M(A3) = M(A1) - M(A2)
121 GOTO 90
130 M(A3) = M(A1) * M(A2)
131 GOTO 90
140 M(A3) = M(A1) / M(A2)
141 GOTO 90
150 READ M(A1)
151 GOTO 90
160 PRINT M(A1)
161 GOTO 90
170 M(A3) = M(A1) [ M(A2)
171 GOTO 90
180 N = A3
181 GOTO 91
190 GOTO 400
200 IF A1 = A2
     THEN
      202
201 GOTO 90
202 N = A3
203 GOTO 91
210 IF A1 < A2
     THEN
      212
211 GOTO 90
212 N = A3
213 GOTO 91
220 IF A1 > A2
     THEN
      222
221 GOTO 90
222 N = A3
223 GOTO 91
400 PRINT
410 PRINT "POST-MORTEM DUMP OF STORAGE"
420 PRINT
430 FOR I = 1 TO 99
440  PRINT I, M(I)
450  NEXT I
460 PRINT
```

```
901 DATA 0105720000
902 DATA 0205730000
903 DATA 0301727358
904 DATA 0406580000
905 DATA 0509000000
906 DATA 0000000000
907 DATA 2
908 DATA 5
```

Into the 80s
Part VI

by Ian R. Sinclair

Once your programs pass the very simple stage, you'll need to present a menu. As the word suggests, a menu is a list of choices for the user. The way it is presented and the way the user makes the choice are all the difference in the world between a program that is a joy and one which is a pain.

The menu should, first, give the user some idea of what the choice is—not just a listing of five numbers! The description needn't elaborate, none of your "sun-ripened section of choice, West Coast subroutine, delectably preserved in quotes"—it isn't that sort of menu—but it must tell all. Figure 1 shows a typical short menu, with choices for keyboard entry, entry of data from cassette, and termination. Termination is important, if the menu is presented several times in the program. There's nothing as infuriating as having to go through several unnecessary steps just to stop a program. The BREAK key can be used to terminate, but it makes sense to construct programs that need little interference.

```
MENU
1. KEYBOARD ENTRY.
2. CASSETTE ENTRY.
3. TERMINATE.
```

Figure 1

Using the command PRINT CHR$(23) just before the menu printout prints it in double-sized characters. The character size can then be returned to normal later with a PRINTCHR$(28), POKE 16445,0, or CLS command.

The simplest way to carry out a menu choice is to type and enter the number of the chosen item. In ordinary BASIC, this would be done as demonstrated in Program Listing 1. The choice is made by typing one of the numbers shown in the menu and ENTERing. Line 30 is an error trap: If you have selected a number that doesn't exist, you are informed and steered back to the menu to try again. That's an important point. If an error trap causes a return with no explanation, the user may not know that there is an error, because there is only a slight flicker on the screen. Showing a message lets the user know that there has been an incorrect entry.

Lines 40 through 70 implement the choice. For each possible menu number, the program is instructed to jump to a different line or to end. At each of the new lines (the examples show 300, 700, 1000), a new section of program must start. This will carry out the action promised by the menu.

That's how a Brand X computer might deal with a menu, but the TRS-80 has a whole lot of tricks up its sleeve. One of the tricks, as far as a menu choice is concerned, is the command ON K GOTO. . . . Program Listing 2 demonstrates this by replacing lines 40 through 60, deleting line 70, and adding a new line, 5000, in Program Listing 1. When you enter a number, it is assigned the variable name K. In the new line 40, the command assumes that K is a number that ranges from one upwards, and it counts the line numbers that are entered between commas. If K is one, it brings you to the first number; if K is two, it brings you to the second; and so on. You must make sure that there are as many line numbers following GOTO as there are choices on the menu. Program Listing 2 is a development of this system; you don't even have to use ENTER. By using the INKEY$ command, whatever number you hit will be assigned to K at once, and your program choice follows. INKEY$ needs a string variable, K$, so that the step K = VAL(K$), or the use of ASC(K$), is needed to convert to the number form, K. Because TRS-80 BASIC has the ELSE command, the conversion can go into the same line. Line 40 makes the error-trapping routine more interesting. If the selection has been correctly made, line 40 is ignored, but a faulty selection causes the words INCORRECT ENTRY to be flashed ten times. This routine also makes use of STRING$.

When using PRINT STRING$, remember a number of characters are printed in a row. The number is the first number specified in the parentheses. The second number is the ASCII code number for the character we want to print. If you can't be bothered to look it up, you can write the character between quotes, like STRING$(25,"*"). In this example, 32 is the ASCII code for a space, so STRING$(15,32) simply replaces the words INCORRECT ENTRY by spaces, deleting the words. Line 50 has the ON K GOTO selection feature, and the line numbers which follow take the program to the routines which are specified on the menu. You can use the same program for different games, because you only have to select a different set of data and instructions for each game.

Alternatives

We don't always want a full menu selection. Sometimes a choice of two is quite enough. There are two methods I use. One is the letter or number method illustrated in Program Listing 3. The choice is between two items, and you are invited to type any number for one or for the other. Once again, we don't use ENTER when you make a choice—though it does give you time

for second thoughts, because we use INKEY$. Line 10 gives the instructions, and line 20 contains the usual INKEY$ instructions. In line 30, we take VAL(K$), which will be zero if K$ is a letter, and use that to decide whether we jump to line 100 (PROCEED) or line 40 (RETURN). It's simple and effective, but it can be phased by typing 0 as a number. A foolproof way makes use of the ASCII codes of letters and numbers and is shown in Program Listing 4.

In line 30 of this program, the first section, K = ASC(K$) finds the ASCII code for the character which has been selected. If this is a number, then its ASCII code is less than 58 and more than 48, and this is sorted out by the second section of line 30. If the character is a letter, its ASCII code is less than 91 (unless you hit SHIFT as well) and more than 64. This also causes a jump. If any other key has been pressed, line 40 registers a mistake and causes a return, after a short delay, to the choice in line 10.

We can go further with the use of INKEY$. Program Listing 5 shows a routine requesting a YES or NO answer directly from the keyboard without using ENTER. It's a development of the YES/NO routine we used in Part IV, with a flashing error message, and a flashing asterisk (which I call a "flashterisk") as a prompt. Just to add bells and whistles, there is a time limit feature—you must type YES or NO quickly to beat the asterisk, or your entry is ignored! These routines, ranging from the simple to the full scale YES/NO are often needed in a program. It is tedious to enter them in each place where needed.

That brings us to subroutines.

Subroutines

A subroutine is a short (or long or middling, but usually short) piece of program which is needed more than once in the course of a main program. It can be called up from different parts of the main program. Calling a subroutine means leaving your main program action and starting the subroutine action. It is implemented by the command GOSUB.

This is another very powerful command, because it saves having to type the same piece of program again and again. To see how it works, look at GOSUB in action in Program Listing 6. Line 10 asks you to type any letter, and line 20 calls up the subroutine in line 100. This consists of the INKEY$ routine. The computer will wait for you to press a key. When that happens, the subroutine returns to the instruction after the place where it was called. In this case, that's the PRINT K$ instruction in line 20. The word LETTER is printed alongside. In line 30, you are asked to type any number, and once again the subroutine is called in line 40. This time the RETURN instruction in line 100 causes the number to be printed with NUMBER alongside because the return is in line 40. Just to be sure, we do it all over again in lines 50 and 60.

See the devilish cunning of it all? It's the same subroutine each time, but it's entered from different parts of the program. It returns to the instruction immediately following the GOSUB which called it. You can have as many GOSUBs as you like, providing each one starts with a line number. You can't call up a subroutine which starts halfway along a line and ends with RETURN.

If you forget the RETURN, the program will crash through, going to the instruction which follows the last line of the subroutine. If there isn't one, the program will end, leaving you wondering what's happened.

If you enter a subroutine incorrectly, for example, forgetting the END in line 70 of Program Listing 6, you'll get an error message in line 100—RG. This means RETURN without GOSUB, because there is a RETURN command, but no GOSUB to call it. There's no record inside the computer of where it should return. It can't return!

You can have a subroutine called from inside another subroutine. This is called nesting, and you can nest subroutines until you run out of memory. Program Listing 7 shows an example of a nested subroutine. The main program asks for a YES/NO answer, and this, in turn, causes a GOSUB to the INKEY$ routine we looked at earlier. This time, however, an error in the typing of YES or NO causes another subroutine to be called, a flashing error subroutine. Because this subroutine can be called from any part of the program, it is available to signal an error later on.

Use subroutines every time a piece of programming is done more than once in a program. The use of INKEY$ is one example. Another is any PRINT routine which is more than a simple PRINT N$ type of command.

A problem that turns up eventually when you start using subroutines is called passing parameters. Look at the simple subroutine in Program Listing 8. It compares two numbers, A and B, to determine which one is larger. This is perfectly straightforward if you have two numbers in the program which are represented by variables A and B. What happens if you haven't, or if you want to compare several sets? This is the problem of passing parameters. Whatever you want to compare has to be converted to the variable numbers A and B, because these are the variables which are used in the subroutine.

In line 10, there is no problem. The numbers are entered directly from the keyboard, and the comparison is made in line 20 by calling the subroutine. In line 30, two words are input, and we compare their lengths by making the variables A and B take the values of the word lengths. Then we call the subroutine. In line 50, two letters are input and their ASCII codes equated to the variables A and B, so the subroutine can be used again to determine the order of the letters. An END command is used just *before* the subroutine to make certain that the subroutine cannot be entered accidentally, but must be called each time it is to be used.

Sometimes parameters have to be passed twice, once when the subroutine is being entered, and again after returning from it. Program Listing 9 illustrates this, using a comparison of numbers which are the tag numbers of strings (the subscripts). In this simple example there is no reason why we should have used L$(N) in the subroutine. If you remember, however, that a subroutine like this would have to be called from several parts of a program—perhaps to sort out string variables—you see it is important to keep the variables used in the subroutine different from those in the main program. Unless we return to the original variables L$(N), the printout in line 40 will be incorrect, because it will show only the original strings. If you have a subroutine which isn't working correctly, it could be that you're not passing parameters!

There's one more useful command which makes use of subroutines. It's a menu command, ON N GOSUB, and it works just like ON N GOTO. Your menu will list choices from one upwards and ask for a choice which is then assigned to the variable SN. When the instruction ON N GOSUB is used, the program will branch to one of a number of subroutines. For example, if there were five menu items, we would need steps such as:

```
100 INPUT N: ON N GOSUB 200,300,400,500,600
```

Input 1, and you go to the subroutine which starts at line 200; input 2, and you go to the subroutine which starts a line 300; and so on. If you want less effort, you can use the INKEY$ answer instead of INPUT. Each subroutine will return to the instruction which follows the ON N GOSUB command, even if the next instruction is on the same line 100.

Neat printing

Messy printing is something that will bug you once you get over the initial thrill of seeing a program work. Professional programs are notable for good, clear, well set out print routines, and there's no reason why yours should look scruffy, especially when you can write a subroutine which can be used more than once, and in different programs that will make your print routines look neat. The main items needing attention are headings and underlining, boxing, and tabulation.

Headings are comparatively easy. The main thing is not to overkill. At the start of a program, it's sensible to have the title displayed in double-sized letters, centered, with underlining, as illustrated in Program Listing 10. If you have another 20 headings the same way though, it will tire the eye. Try grading your headings in order of importance, with double-sized letters used once, apart from a flashing error warning. The next important headings can use double-spaced letters with underlines, such as in line 30 of Program Listing 10. The least important headings can be inset (using TAB(10)) and not underlined, but with a one line gap underneath. There is no reason why

you should follow that scheme, but it does illustrate what I mean. To match with the headings, print menus are in the same style.

To avoid looking at a set of instructions each time you run a program, contain the instructions in a subroutine. They can then be consulted at the start of a program if needed, but skipped if not.

A further refinement—if your program demands a lot of memory space, delete the instruction lines automatically if they are not needed. The DELETE command will run just as efficiently as a program instruction, as it does in direct command mode. Program Listing 11 shows an example of this.

Boxing is another way to draw attention to something. This can be effective when a question is asked, and an answer has to be typed—look at Program Listing 12, and run it to see what happens. The box is drawn in lines 20 and 30. In line 20 we pick the X-values of the ends of the box, and draw lines down, making the box three print lines deep. In line 30 we draw them across to complete the box. To program these effects, you need to use the video map on page E/1 of your manual. I clip a piece of tracing paper over the video map and draw the shapes I want on top of the paper. I can then see what has to be SET, to make the shape. It's easy if you want only straight horizontal or vertical lines. You can then use one FOR-NEXT loop for the Ys and another for the Xs.

Boxing can create some interesting effects. One is illustrated by adding the new line 50 shown in Program Listing 13. Each character of the name is peeled off by using the MID$ instruction, and at the same time part of the floor of the box is reset. The effect is of letters dropping down and knocking holes in the box, and it adds a bit of interest to what might be only a dull INPUT. Remember though, once per program is enough for these tricks.

Tabulation is one of the things that can make a video screen or paper printout look really professional. You may not feel your programs need neat tabulation, but who knows? Take a look at Program Listing 14, which uses string tabulations to round off the game from Part III. This is an easy one, because four columns can be set by using the comma as a delimiter. We then use a FOR-NEXT loop to print out the items, again using the commas as delimiters. Another way of creating neat tabulation is to make use of the TAB instruction. It has been used in Program Listing 15 to create a neat display of 90 random numbers. Line 10 sets up an array of 90 random numbers of two digits. The print tabulation is in line 20, using two FOR-NEXT loops. The first, FOR X = 1 TO 90 STEP 9 sets up ten lines of numbers, and FOR Y = 0 TO 8 creates the nine number positions across each line.

Program Listing 16 shows the part of the routine which is of interest to us. Since this is a money table, we assume that the quantities are dollars and cents, and there are two figures after each decimal point. That means that if the last cents figures are lined up, the decimal points must also be lined up,

and we can easily line up the right-hand side by using TAB and LEN. The figures are entered in line 10, and the variable used for the total T is set to zero. In line 30, we set up another FOR-NEXT loop in which we calculate the total (T = T + Z(N)) and convert the quantity Z(N) to a string so that we can use LEN on it. In line 40, we print the value Z(N) at the tab position LEN(Z$), which starts before TAB(30). This spacing should be just right to get the end figures of the quantity on the TAB(30) position.

What do you do if someone enters a number not having the correct number of figures after the decimal point? The obvious answer (to me, anyhow) is to pack the number with zeros until it has two figures after the point. The question is—how? Program Listing 17 will do just that. We have a new entry procedure here, which turns each number into a string, Z$, and then tests Z$ to find where it has a decimal point. This is done by finding the length of Z$ and examining each character in turn, using the FOR-NEXT loop to see if a decimal point is present—ASCII code 46. If the figure, converted into a string, has two digits after the decimal point, it will be detected in line 30. If Z$ = 142.64 we will jump out of the loop in line 20 at K = 4, because the decimal point is the fourth character along from the start. The total number of characters is 6, so Z = K − 2, and we can jump to line 60 to print the amount.

Line 40 detects a figure with one digit after the decimal point. When this happens, Z − K = 1, so we can add a zero to pad the number string out. Finally, line 50 sorts out the last possibilities. If the number has been written with a decimal point but nothing after it, a pair of zeroes will be added, and if there is no decimal point (so that K will have taken a value of L + 1 before stopping the loop, and Z − K = − 1), then a decimal point and two zeroes are added.

This program applies to money quantities, but the techniques can be adapted for anything else where you need to recognize a feature and line up on it.

Tape on Tap

In Part II we looked at the CSAVE and CLOAD procedures for recording and replaying programs, to avoid the tedious task of having to key in a program each time you switch the computer on. You've probably discovered other chunks of information that you don't want to have to enter each time. If you have a home finance program which you use once a week, you certainly don't want to spend the last week of the year reentering all the data for the 51 previous weeks. On the other hand, you don't want to keep the computer running all year, so you can enter financial data once a week. You need to record the data once a week so that it can be recalled.

Some programs need more data than the computer can hold, though it

may not be needed all at once. In these cases, data has to be stored on cassette or some other storage system. A pack of recorded data is called a file, and data filing is an important and interesting topic. Cassette data files are called serial files—you start recording at one end of the tape, and you keep on until you're through or you hit the other end. There's no way you can automatically pick a piece out of the middle of the taped data without reading everything that's gone before, unless you note the tape counter readings of recorded sections. This problem drives most people to use disks because a disk system comes with an operating program (the Disk Operating System, or DOS). It does the file-finding for you. Disk filing isn't all sweetness and light though. It's my opinion that most nonprofessional users don't need disk, especially since there is an alternative, the Exatron Stringy Floppy™.

Back to cassette files. There's a record command and a replay command. The record command is PRINT# – 1, and the replay command is INPUT# – 1. When you play back, two asterisks appear, but they don't blink. The # – 1 means that we have only one cassette recorder on line. If you have an expansion interface (which disqualifies you as a beginner), you can run two cassettes, # – 1, and # – 2. Since most people buy expansion interfaces to avoid cassettes, we'll stick to the # – 1 channel, however, which uses the normal five-pin cassette connector at the back of the TRS-80.

There's a world of difference between the CSAVE and CLOAD commands and the PRINT# – 1, and INPUT# – 1 commands. When you CSAVE a program, the listing is saved. You don't have to do anything special to ensure this. Similarly, when you use CLOAD, you load in the whole program. PRINT# – 1 and INPUT# – 1 are different. You have to say if you are recording or replaying a string or a number, and the size is then restricted. The maximum safe size for one record operation is 248 characters; you can send out 255, but you can only get back 248. In addition, you have to say what you are recording, and *no commas must appear, even within quotes*, in the string which is recorded.

Suppose you have two strings, L$ and S$, and two numbers, N and J, which you want to record. Your recording command will look something like:

```
100 PRINT# – 1, L$,S$,N,J
```

When that instruction comes along, you must be prepared with a cassette ready to record, and the record/play keys pressed. Usually we have a "hit any key to start" step just before the recording stage, as shown in Program Listing 18. The PRINT# – 1 instruction in line 40 will record these items on the tape, along with a leader and a brief trailer (end byte). Each PRINT# – command causes the leader to be recorded, followed by the data, then the trailer, so that quite a lot of tape will be used even if there is only one byte of data to be recorded.

To replay the data, you need a section which contains an INPUT# – 1 command with the same arrangement of strings and numbers as the PRINT# – 1 command. The variable names don't have to be the same, but the order and number of the variable data must be. If we want to replay the data recorded by the PRINT – 1 command used in the previous example, we could use an instruction such as:

```
INPUT# - 1, A$,B$,C,D
```

This uses different variable names, but the arrangement is identical—two strings followed by two numbers. Any other order, or a differing number of string or number variables, will cause an error message, FD (faulty data) if the sequence is wrong, or OD (out of data) if you have asked for more data in the INPUT command than was recorded. It's not a bad idea to use different variable names on the replay.

This is very straightforward stuff, but as it is (and the manual isn't helpful on this point), it requires time-consuming routines. It uses a lot of tape. If you set up a loop which looks something like:

```
200 for J = 1 to 100: PRINT# - 1,N(J):NEXT
```

you'll wait a long time while it records, because each step in the FOR-NEXT loop starts a new recording with leader and trailer. This line will cause 100 recording runs!

Pack it Close

Since you probably bought a computer to save time, this lengthy procedure is useless. All would be well if we could just use a line like 300 PRINT – 1, FOR J = 1 TO 100:N(J):NEXT—but we can't. The problem is to pack the data so that 240 bytes or so can be recorded in one chunk. I haven't seen this topic discussed very much in magazine articles, but it's one I've spent time on. It could be that this will help even hardened, old time operators.

The solution is simple, if the data consists of numbers or strings which are the same length. If they are numbers, convert them into strings, using the STR$(number) command. Remember, this packs strings into long strings, using the + (concatenate) string action, and records the long string. Program Listing 19 shows what has to be done. Line 10 sets up a FOR-NEXT loop to input 50 numbers of four digits each, and the CLEAR statement prepares for this. Each number is converted into a string as it is entered, and the length is tested to make sure you don't cheat. These numbers can come from any part of the program. The packing routine is in line 40. S$ was initialized as a blank string in line 10, and it now has the number string tagged on. After three strings, for example, 1234, 5678, and 9012, the string S$ is 123456789012. This string increases until all 50 numbers have been joined, and S$ is 200 characters. The long string is recorded. There will be a leader and a trailer recorded with it, and you have saved a lot of recording time.

If the number existed in the form of N(J) before the packing step, you will need a FOR-NEXT loop which converts each number into a string and then packs it. Make sure that S$ is set to blank ("") before the FOR-NEXT loop which packs it. Otherwise you can get some peculiar results when you do the routine more than once.

Replaying a packed string is easy, provided you know how it was packed. In this example, we used data in four-digit units, 50 to a string. Our replay procedure looks something like Program Listing 20. Lines 500–520 are the usual replaying procedure, rewound to the correct place, ready for replay. At 520, the 200-byte string which we've labelled L$ will read in from the cassette. Converting this into the form we need, 50 sets of numbers, is done in line 530. The FOR-NEXT loop sets the number as 50, and the expression L(N) = VAL(MID$(L$,4*N – 3,4)) gets the groups. When N = 1, we try to find VAL(MID$(L$,4*1 – 3,4) which is the value of the group of four characters starting with character 1. That's the first set of four. When N = 2, 4*N – 3 is 5, so we read another four starting with character five. That's the second set of four.

The key to this is the formula 4*N – 3 which we've used to find the first character from 1 to 50. Whatever number of digits you use for a group, it is always similar; it's the number of coded digits multiplied by N, with one less than the group number subtracted; if you are dealing with seven-character groups, the formula would be 7*N – 6. This style of packing and unpacking can make cassette files more efficient than the Level II manual suggests. It can even delay abandoning cassettes!

Suppose that the items you record are not in convenient groups of four or whatever? One answer is to pack them so that they are a standard length. Suppose the items are single-precision numbers with up to six digits. There's no reason why any number of less than six shouldn't be packed with blanks up to six-digit length, using something like Program Listing 21. The key part of the routine is in line 20:

```
N$ = STRING$(7-LEN(N$),32) + N$
```

If N$ is 25.2, then the length of the string is five characters, because the STR$ conversion always adds a leading blank. The instruction is to form a number of blanks (ASCII 32) equal to 7 – 5, and add these to N$, making N$ two blanks longer. When this lot is unpacked, the VAL command will simply remove these blanks again.

Maybe you're hard to please, and you want to pack together strings of different lengths. You'll object to packing your valuable tape with blanks, which can happen if some are one or two characters and others 20 or more. There are two solutions in BASIC which I use (and others in machine code). I'll describe the simplest of the BASIC methods here—it's rather slow, but it works well. This routine depends on the use of ASCII character 128. It is a

blank, like ASCII 32, but with a difference. ASCII 32 is what you get when you hit the space bar on the keyboard, but ASCII 128 never gets entered from the keyboard, and the computer recognizes it as a different character. If we pack strings with 128 between them, we should be able to unpack them by scanning the replayed string and looking for the 128 code number. This is what makes the replay slow, because all the replayed characters have to be checked. There is an enormous saving in time, compared with recording each string. The speed of the routine doesn't matter if the strings are displayed on the video screen. If anything, we'll probably want to slow things down.

Program Listing 22 shows the routines. The packing is fairly straightforward, and the long string is formed with a CHR$(128) between each added string. Make sure that the total number N is recorded to make playback easy. Another addition is the string length detecting routine in line 570. This ensures that the string does not become too long to record, because you don't know how many you can pack. If one more string makes the total too long, it is recorded, then reset to zero so that packing can continue.

The unpacking routine examines each character of a replayed long string until a 128 appears. The assembled string is given a subscript number, the number is incremented, and the unpacking routine continues. The end comes when the subscript number equals the number of strings recorded, or when an end of data code is detected. I've opted for a recorded number in this example.

Routines like these convert cassette data files from rather useless curiosities into reasonable methods of storing and replaying data. The high-speed methods using machine code (with routines built into the TRS-80 ROM) can be impressive. One warning—always make a backup cassette of valuable data, just as you make a backup of a valuable program. It's worthwhile to put a special routine in your programs to do this, such as in Program Listing 23.

Program Listing 1

```
10 CLS:DEFINT K,N
20 PRINTCHR$(23)TAB(15)"MENU":PRINTTAB(15)"****":PRINTT
    AB(1)"1. TO ENTER NEW DATA":PRINTTAB(1)"2. TO READ
     EXISTING DATA":PRINTTAB(1)"3. TO ENTER DATA FROM
    TAPE":PRINTTAB(1)"4. TO TERMINATE PROGRAM":PRINT"P
    LEASE CHOOSE BY NUMBER"
30 INPUT K:IF K<1 OR K>4 THEN CLS:PRINT"MISTAKE - PLEAS
    E TRY AGAIN":FOR N=1TO500:NEXT:GOTO20
40 IF K=1 THEN 300
50 IF K=2 THEN 700
60 IF K=3 THEN 1000
70 IF K=4 THEN CLS:END
300 CLS:PRINT"DATA ENTRY STARTS HERE":STOP
700 CLS:PRINT"DATA READ SECTION STARTS HERE":STOP
1000 CLS:PRINT"TAPE ENTRY SECTION STARTS HERE":STOP
```

Program Listing 2

```
10 DEFINT K,N
20 CLS:PRINTCHR$(23)TAB(15)"MENU":PRINTTAB(15)"####":PR
    INTTAB(2)"1. GAME OF GENDER":PRINTTAB(2)"2. GAME O
    F GROUPS":PRINTTAB(2)"3. GAME OF YOUNG":PRINTTAB(2
    )"4. TERMINATE":PRINT:PRINTTAB(2)"PLEASE SELECT BY
     NUMBER"
30 K$=INKEY$:IF K$="" THEN 30 ELSE K=VAL(K$)
40 IF K<1 OR K>4 THEN CLS:FOR N=1TO10:PRINT@473,"INCORR
    ECT ENTRY":FOR Z=1TO20:NEXT Z:PRINT@473,STRING$(15
    ,32):FOR Z=1TO20:NEXT Z:NEXT N:GOTO20
50 ON K GOTO 300,700,1000,5000
60 END
300 CLS:PRINTTAB(25)"GAME OFGENDER":STOP
700 CLS:PRINTTAB(25)"GAME OF GROUPS":STOP
1000 CLS:PRINTTAB(25)"GAME OF YOUNG":STOP
5000 CLS:END
```

Program Listing 3

```
10 PRINT"HIT ANY LETTER TO PROCEED, ANY NUMBER TO RETUR
    N"
20 K$=INKEY$:IF K$="" THEN 20
30 IF VAL(K$)=0 THEN 100 ELSE40
40 CLS:PRINT"RETURN PROGRAM STARTS HERE":STOP
100 CLS:PRINT"PROCEED PROGRAM STARTS HERE":STOP
```

Program Listing 4

```
10 CLS:PRINT"HIT ANY LETTER TO PROCEED, ANY NUMBER TO R
    ETURN"
20 K$=INKEY$:IF K$="" THEN 20
30 K=ASC(K$):IF K>48 AND K<58 THEN 50 ELSE IF K>64 AND
    K<91 THEN 100
40 CLS:PRINT@480, "MISTAKE":FOR N=1TO500:NEXT:GOTO10
50 PRINT "RETURN PROGRAM":STOP
100 PRINT "PROCEED PROGRAM":STOP
```

Program Listing 5

```
1000 CLS:A$=""
1010 K$=INKEY$:IF K$=""THEN 1200 ELSE PRINT K$;
1020 A$=A$+K$:IF LEN(A$)<2 THEN 1010
1030 IF LEN(A$)=2 AND A$="NO" THEN M=2:GOTO2000
1040 IF LEN(A$)=3 AND A$="YES" THEN M=1:GOTO2000
1050 IF LEN(A$)=2 GOTO1010 ELSE F$="MISTAKE":GOTO1500
1060 END
1200 PRINT@1,"*":FOR Z=1TO30:NEXT Z:PRINT@1," ":FOR Z=1
     TO30:NEXT Z:GOTO1010
1500 CLS:PRINTCHR$(23):FOR I=1TO15:PRINT@470,F$:FOR J=1
     TO20:NEXT J:PRINT@470,STRING$(20,32):FOR J=1TO20:N
     EXT J:NEXT I:PRINTCHR$(28):GOTO1000
2000 IF M=1 THEN CLS:PRINT"THE 'YES' PROGRAM FOLLOWS":E
     LSE CLS:PRINT "THE 'NO' PROGRAM FOLLOWS"
```

Program Listing 6

```
10 CLS:PRINT "TYPE ANY LETTER"
20 GOSUB 100:PRINT K$;"  (LETTER)"
30 PRINT:PRINT"TYPE ANY NUMBER"
40 GOSUB 100:PRINT K$;"  (NUMBER)"
50 PRINT:PRINT"NOW TRY ANY KEY"
60 GOSUB 100:PRINT "YOU CONFUSED ME"
70 END
100 K$=INKEY$:IF K$="" THEN 100 ELSE RETURN
110 END
```

Program Listing 7

```
10 CLS:PRINT"PLEASE TYPE YES OR NO (DON'T USE ENTER)":G
     OSUB 1000:PRINT :CLS:PRINT"YOUR CHOICE WAS ";M
20 END
1000 A$=""
1010 K$=INKEY$:IF K$="" THEN 1010 ELSE PRINT K$;
1020 A$=A$+K$:IF LEN(A$)<2 THEN 1010
1030 IF LEN(A$)=2 AND A$="NO" THEN M=2:RETURN
1040 IF LEN(A$)=3 AND A$="YES" THEN M=1:RETURN
1050 IF LEN(A$)=2 THEN 1010 ELSE F$="MISTAKE":GOSUB 120
     0:GOTO10
1200 CLS:PRINTCHR$(23):FOR I=1TO15:PRINT@470,F$:FOR J=1
     TO20:NEXT J:PRINT@470,STRING$(20,32):FOR J=1TO20:N
     EXT J:NEXT I:PRINTCHR$(28):RETURN
```

Program Listing 8

```
10 CLS:INPUT "TWO NUMBERS,PLEASE";A,B
20 GOSUB510
30 INPUT"TWO WORDS, PLEASE"; N$,L$
40 A=LEN(N$):B=LEN(L$):GOSUB 510
50 INPUT "TWO LETTERS, PLEASE";A$,B$
60 A=ASC(A$):B=ASC(B$):GOSUB 510
70 END
500 END
510 IF A>B THEN CLS:PRINT "FIRST IS LARGER"
520 IF A=B THEN CLS:PRINT "THEY ARE EQUAL"
530 IF A<B THEN CLS:PRINT "SECOND IS LARGER"
540 RETURN
```

Program Listing 9

```
10 REM YOU WOULD PLACE A DIM STATEMENT HERE
20 FOR N=1TO6:READ X(N),L$(N):NEXT
30 FOR N=1TO6 STEP2:A=X(N):B=X(N+1):Y$(N)=L$(N):Y$(N+1)
    =L$(N+1):GOSUB 200:L$(N)=Y$(N):L$(N+1)=Y$(N+1)
40 PRINT X(N);;;Y$(N);TAB(30)X(N+1);;Y$(N+1):NEXT
190 END
200 IF A>B THEN Z$=Y$(N):Y$(N)=Y$(N+1):Y$(N+1)=Z$
210 IF A=B THEN PRINT "EQUAL";
220 RETURN
400 DATA 2,"THUMB",1,"FINGER",7,"TOE",14,"FOOT",8,"JAW"
    ,8,"EAR"
410 END
```

Program Listing 10

```
10 CLS:PRINT@344,CHR$(23)"HEADING":PRINTTAB(12)STRING$(
    7,48)
20 FOR N=1TO1200:NEXT:PRINT CHR$(28):PRINT TAB(13);
30 L$="SUBHEADING":FOR N=1TO LEN(L$):PRINT MID$(L$,N,1)
    ;"   ";:NEXT:PRINT:PRINTTAB(13)STRING$(37,48)
40 PRINT:PRINTTAB(6)"SUB-HEADING!":PRINT:PRINTTAB(2)"TH
    IS GIVES A REASONABLY NEAT APPEARANCE"
```

Program Listing 11

```
10 CLS:PRINT@3,"DO YOU NEED INSTRUCTIONS?":GOSUB 1000:I
    F M=1 THEN GOSUB 5000 ELSE IF M=2 THEN DELETE 5000
    -5020
20 PRINT "NEXT STEP"
30 STOP
1000 REM THE YES/NO SUBROUTINE GOES HERE
1010 RETURN
5000 PRINTTAB(26)"INSTRUCTIONS":PRINTTAB(26)STRING$(12,
    48):PRINT
5010 PRINTTAB(2) "THE OPERATING INSTRUCTIONS GO HERE"
5020 RETURN
```

Program Listing 12

```
10 CLS:PRINT@325,"WHAT IS YOUR NAME?"
20 FOR Y=18 TO 26:SET(32,Y):SET(92,Y):NEXT
30 FOR X=32 TO 92:SET(X,18):SET(X,26):NEXT
40 PRINT@465,"";:INPUT N$
50 PRINT@710,"YOUR NAME IS ";N$;" ,HUH?"
60 END
```

Program Listing 13

```
10 CLS:PRINT@325,"WHAT IS YOUR NAME?"
20 FOR Y=18 TO 26:SET(32,Y):SET(92,Y):NEXT
30 FOR X=32 TO 92:SET(X,18):SET(X,26):NEXT
40 PRINT@465,"";:INPUT N$:FOR X=32 TO 92:SET(X,24):NEXT
50 FOR L=1 TO LEN(N$):RESET (36+2*L,24):PRINT@657+L,MID
    $(N$,L,1):FOR N=1TO 150:NEXT N:NEXT L
60 END
```

Program Listing 14

```
10 FOR N=1TO4 :READ A$(N),F$(N),Y$(N),G$(N):NEXT
20 CLS:PRINT "MALE","FEMALE","YOUNG","GROUP":PRINT:PRIN
    T
30 FOR N=1TO4:PRINT A$(N),F$(N),Y$(N),G$(N):NEXT
40 DATA "GANDER","GOOSE","GOSLING","GAGGLE","BULL","COW
    ","CALF","HERD","RAM","EWE","LAMB","FLOCK","DOG","
    BITCH","PUPPY","PACK"
```

Program Listing 15

```
10 DIM N(100):FOR L=1TO90:N(L)=RND(99):NEXT
20 CLS:FOR X=1TO90 STEP 9:FOR Y=0TO8:PRINTTAB(Y*6+4)N(X
    +Y);:NEXT Y:PRINT:NEXT X
```

Program Listing 16

```
10 CLEAR200:FOR N=1 TO 8:INPUT "CASH AMOUNT"; Z(N):NEXT
    :T=0
20 PRINT "CASH SUMS"
30 FOR N=1TO8:T=T+Z(N):Z$(N)=STR$(Z(N))
40 PRINTTAB(30 - LEN(Z$(N)))Z$(N):NEXT
50 PRINT:PRINTTAB(7)"TOTAL IS :-"TAB(30 - LEN(STR$(T)))
    T
```

Program Listing 17

```
10 CLEAR 200:FOR N=1TO8:INPUT "CASH AMOUNT";Z$(N):Z=LEN
    (Z$(N))
20 FOR K=1 TO Z:IF MID$(Z$(N),K,1)<>"." THEN NEXT K
30 IF Z-K=2 THEN 55
40 IF Z-K =1 THEN Z$(N)=Z$(N)+"0":GOTO55
50 IF Z-K=0 THEN Z$(N)=Z$(N)+"00" ELSE Z$(N)=Z$(N)+".00
    "
55 NEXT N
60 T=0:CLS:PRINT"CASH SUMS"
70 FOR N=1TO8:T=T+VAL(Z$(N))
80 PRINTTAB(30-LEN(Z$(N)))Z$(N):NEXT
90 PRINT:PRINTTAB(7)"TOTAL IS :- "TAB(30-LEN(STR$(T)))T
100 END
```

Program Listing 18

```
10 INPUT "TWO WORDS, PLEASE";L$,S$
20 INPUT"...AND NOW TWO NUMBERS";N,J
30 CLS:PRINT@135,"PREPARE FOR RECORDING DATA,PLEASE":PR
    INT:PRINT"PRESS ANY KEY TO START RECORDER"
40 K$=INKEY$:IF K$="" THEN 40
50 PRINT#-1,L$,S$,N,J
60 CLS:PRINT "RECORDING COMPLETE - PRESS STOP KEY ON RE
    CORDER"
70 END
```

Program Listing 19

```
10 CLEAR300:S$="":FOR J=1TO50
20 INPUT "ENTER A FOUR-FIGURE NUMBER,PLEASE";N:N$=RIGHT
    $(STR$(N),4)
30 S$=S$+N$:NEXT
40 CLS:PRINT@330,"PLEASE PREPARE FOR RECORDING - PRESS
    ANY KEY TO START"
50 K$=INKEY$:IF K$="" THEN 50
60 PRINT#-1,S$
70 CLS:PRINT"RECORDING COMPLETE":PRINT "S$ IS ";S$
```

Program Listing 20

```
500 DIM L(51):PRINT"PLEASE PREPARE FOR REPLAY. HIT ANY
     KEY WHEN READY":L$="":N=0
510 K$=INKEY$:IF K$="" THEN 510
520 INPUT#-1,L$
530 FOR N=1TO50:L(N)=VAL(MID$(L$,4*N-3,4)):NEXT
540 PRINT"DATA READY - ":FOR Z=1TO500:NEXT:FOR N=1TO50
     STEP 10:FOR X=0TO9:PRINT L(N+X);:NEXT X:PRINT:NEXT
      N
550 END
```

Program Listing 21

```
10 S$="":FOR J=1 TO 30:PRINTJ".";:INPUT N
20 N$=STR$(N):N$=STRING$(7-LEN(N$),32)+N$:S$=S$+N$:NEXT
30 REM DATA IS NOW PACKED
40 PRINT:PRINT"PACKED DATA - ";S$
50 REM NOW RECORD IT!
```

Program Listing 22

```
500 CLS:PRINT"PREPARE A CASSETTE FOR A TAPE FILE"
510 PRINT"NOTE THE STARTING POINT ON THE TAPE COUNTER,
     AND PRESS THE PLAY AND RECORD KEYS"
520 PRINT"PRESS ENTER WHEN READY"
530 INPUT X:CLS:PRINTTAB(21)"RECORDING...PLEASE WAIT"
540 PRINT #-1,I: REM I IS THE NUMBER OF ITEMS
550 A$=""
560 FOR N=1 TO I:A$=A$+L$(N)+CHR$(128)
570 IF LEN(A$)+LEN(L$(N+1)<245 THEN 590
580 PRINT#-1,A$:A$=""
590 NEXT N:PRINT #-1,A$
600 CLS:PRINT" RECORDING FINISHED. PRESS ENTER TO RETUR
     N TO MENU":REM NEED A RETURN TO MENU ROUTINE HERE
610 STOP: REM REPLAY ROUTINE STARTS HERE
620 CLS:PRINT@336,"PREPARE THE DATA TAPE FOR REPLAY"
630 PRINTTAB(13)"PRESS PLAY KEY; WHEN READY PRESS ENTER
     "
640 INPUT X:CLS:PRINTTAB(19)"ENTERING DATA, PLEASE WAIT
     ":X=1
650 INPUT#-1,I
660 INPUT#-1,A$:FOR N=1TO245:B$=MID$(A$,N,1)
670 IF B$<>CHR$(128) THEN L$(X)=L$(X)+B$:GOTO690
680 X=X+1
```

Program continued

```
690 NEXT N:IF X<I GOTO660
700 CLS:PRINTTAB(26)"DATA ENTERED."
710 REM NOW YOU DISPLAY, OR OTHERWISE USE DATA
```

Program Listing 23

```
1000 T=0: REM DATA MUST BE IN SINGLE STRING FORM. THE M
     ESSAGE WILL ALREADY HAVE PRINTED THAT RECORDER IS
     TO BE MADE READY, ENTER PRESSED TO START RECORDING
1010 PRINT#-1, A$:IF T=1 THEN 1040
1020 PRINT"NOW PREPARE BACKUP CASSETTE":T=1
1030 CLS:PRINT"PRESS ENTER WHEN READY TO RECORD BACKUP"
     :INPUT X:GOTO1010
1040 CLS:PRINT "RECORDING COMPLETE":REM PROGRAM THEN PR
     OCEEDS
```

TUTORIAL

Into the 80s
Part VII

by Ian R. Sinclair

By now, you've lost your beginner's status, and it's time to look at a few items which had to be left aside earlier. The first of these is program planning. Many BASIC programs grow untidily and haphazardly from an idea or from another program. We've all looked at someone else's program and thought, "Hey—I could really make something out of this." After a lot of work you can have a program that pleases you, but it's what I call a Stein program—Frank N. Stein—made from bits and pieces, and full of GOTOs.

When you start programming, you're glad to write a program that works, and you don't really care about how it was planned and what it looks like. You should now start to care about these points, because there is a considerable saving in time that can be made.

I don't have much time to spend in front of the computer. Most of my program work has to be done in other places, at lunch breaks, where and when I have odd moments. Because computer time is precious, I don't want to spend it sorting out syntax errors, NEXT without FOR, and other needless errors. This is particularly important on the Level II TRS-80, because every time you edit, you lose any variable values in the program, so the whole program must be run again from the start. It makes sense, therefore, to have all syntax errors sorted out before you enter a program.

Why *syntax* in particular? When the TRS-80 detects a syntax error, which might be very trivial, it hangs up the run, displays the SN error message, and then enters edit mode automatically, with the offending line number displayed and waiting for you to edit it. If you make any attempt to edit, even L for LIST, you reset all your variables. On some programs this wouldn't matter, but if your program involves reading in data from tape or entering a number of items from the keyboard, you won't want to lose it if there's any way of avoiding it. Make a note of the line number and then press the RESET button; this takes you out of the EDIT mode without losing the variables. Alternatively, press Q for Quit.

You can now type GOTO (next line) and processing will take up from there, unless there is a NEXT whose FOR was in the previous line. If the line you lost had an important command in it, substitute with a direct command. For example, if the line read 510 PRRINT"VALUE IS";V: N=V↑2+2*C-6*L, and it hung up because of the double R in PRINT (when will you do something about that key bounce?), press RESET, type N=V↑2+2*C-6*L ENTER, and then type GOTO 520, assuming that next line is 520, and ENTER again. Your program should then continue.

Practically all the commands of the TRS-80 can be used either from the keyboard directly or within a program with the same effect.

You can spend a lot of time at the keyboard sorting out flaws which never should have gotten that far. Program planning should make your keyboard hours more productive. A program should start with an outline plan of what you want it to do. If it's a game program, you need to consider what the strategy of the game is to be and write down the rules. This is the hardest part of any game program, and it's why there are several thousand versions of "Hangman." If you start with simple, established rules, you've saved yourself months of effort. You can't start programming until you know what you need to program.

Once you have a clear idea of what the program is supposed to do, write it down. It's only too easy to make a lot of alterations to a program, which will leave you at the next session wondering what it was you wanted to do, and why you did it. Lots of professional programmers use flowcharts, and flowcharting is urged on every trainee programmer. I dislike flowcharts. They complicate rather than simplify for me, causing too much visual clutter. You can find plenty of reading matter about flowcharts elsewhere—I'm going to describe how to work without them. To be fair, flowcharts can be very useful when you are working in other programming languages, but I feel that they aren't really appropriate to BASIC.

I start by writing down what I expect the program to do—at what stages I need to put in information and at what stages I expect to see information on the screen (or the printer). This is my equivalent of flowcharting, but in words rather than in pictures. Once I'm sure what I want the program to do, I sit down with a stack of paper. As I use a sheet, I title it and give it a page number. Next, I design any menu stages. I also note what is going to happen when each choice is made.

Construction

The next step is program construction. I usually go for a very short (10–20 line numbers) main program, with the choices arranged as subroutines, so that I can alter them as much as I want to later. Program Listing 1 shows an example of this—each part which might need changing is a subroutine, and the main program consists of only five lines. All the INKEY$ steps, YES/NO decisions, and so on, are left as subroutines, since they can be standard subroutines which are used in several programs. I keep a tape of all my subroutines and run that in as a starter for any program I am entering.

If the title is short, the method of underlining I've shown in Program Listing 1 is quicker than using STRING$. If the complete program is long, it's a good idea to delete the instruction lines if instructions are not needed. This can be done immediately after the decision step by using the fact that a NO

answer returns M = 2. A line such as 100 IF M = 2 THEN DELETE 8000-8100 will delete instruction lines 8000 through 8100. This leaves you free to design your instructions after the program is running as you want it. Any alterations you make in the instructions won't affect the main program. If the instruction lines are lines 20 through 120, adding more instructions will have to be done carefully. In addition, it's difficult to follow what the program does when it gets cluttered with extra lines.

The rest of the program is designed in the same way, with a menu display followed by the INKEY$ routine letting you choose an item. This subroutine would normally include any error-trapping you needed. Line 40 is the menu subroutine crossroads. Line 50 is needed because after any of these subroutines has been used, the program will return at line 50. We have to offer the choice of a return to menu or ending the program.

Not every program can be put into this form; there are programs which need no menu choices, and which use very few subroutines. Once the main program has been written you can start designing the subroutines. Each of these should be treated just like a main program.

Watch Your Variables

Each time you make use of a variable, N, A$, or whatever it is, write down what you use it for and in what lines it's used. For example, it's easy to get into the habit of using N in FOR-NEXT loops. If you use N for anything else within a loop, you will wreck the loop. If you start a loop as 2400 FOR N = 1 TO 200:INPUT N$(N) and follow it up with something like 2420 PRINT-CHR$(23)L$(N):FOR N = 1TO1500:NEXT and then several lines later you have 2450 NEXT, don't be surprised if odd things happen.

At 2400, N will take the value 1, and you input the first string. At 2420, this string is printed, and a delay loop is used to keep the large letters on the screen for a time. The trouble is that the delay loop also uses N, so that at the end of the delay loop, N will be set to 1501. At line 2450, the NEXT command will find that N is 1501, much greater than the jump-out-of-loop value which was set in line 2400, so the loop stops. You could spend a lot of time wondering why only one value ever got itself input and displayed, especially if you save memory by writing long lines with the variables buried deep inside.

Also note how each subroutine uses variables from the main program or from other subroutines, and what it does with them. For example, if you have a set of strings stored as array L$(N), and these are used by a subroutine and changed in the subroutine, make a note of this. If you need to use the unchanged value in another subroutine, you will have to use a different variable name for the changed value.

After using this method of constructing programs for some time, you'll have a good stock of useful subroutines. Some of these (neat printing

routines, tabulated displays, record and replay of packed data, etc.) are likely to be used in every program you write. Keep them together on a cassette, with a note of what they do and what line numbers are used. Make sure you also note what variables each subroutine uses, what variables have to be passed to it, and what variables it passes back. If you write programs which use many subroutines, then you can update and improve them easily. Got a tape replay subroutine which is too slow? Some day you'll come across a faster one, and you'll be able to rewrite it in subroutine form and use it to replace the old one. Even if these new routines need more lines, the subroutine methods allow you to leave plenty of space so there's no need to try to shoehorn a new routine in between lines 40 and 50, for example.

PEEK and POKE

Everything that goes on inside the TRS-80 involves the use of machine code. Each command used in BASIC calls up a machine-code subroutine which in turn calls other subroutines to do the work. The BASIC used by the TRS-80 is *interpreted* BASIC, which means that when you RUN, each command calls up the machine-code subroutines one by one as the program progresses. This is a lengthy and clumsy way of using BASIC, and mainframe machines use a program called a compiler, which converts all the BASIC commands of a program into one large machine-code program, rather than operating piece by piece. Compiled BASIC runs a lot faster, but it's not easy to edit, which is why it's not used much in microcomputers. A good machine-language program runs a lot faster and takes up much less memory space than the equivalent BASIC program, which is why so many long programs are written in machine code.

A machine-language program consists entirely of numbers between 0 and 255. A lot of books and articles show these numbers in the hexadecimal scale, which is based on sixteen, as compared to the normal scale based on ten. Hexadecimal numbers use the letters A through F to represent the decimal numbers 10 through 15, and they make any machine-language program look highly exotic until you get accustomed to them.

The use of hexadecimal codes is a hangover from early microprocessor units, and the TRS-80 displays every number in ordinary decimal form. It therefore seems pointless to keep converting numbers into hexadecimal and back again just to represent machine-code programs, which the computer converts into binary code anyway. A lot of information in the TRS-80 codes comes in a mixture of hex and decimal. Conversions always cause mistakes, so I work entirely in decimal unless I am writing new machine-code programs.

The Program Line

The PEEK command lets you find code stored at any memory location in-

side the computer. Using PEEK does not alter any of the codes, so you can PEEK to your heart's content. One very instructive PEEK is at a program instruction line, and Program Listing 2 lets you do just that. The number 17129 is the address number of the first byte of free memory into which BASIC programs are entered. When you PEEK there, you are looking at the first code of any program that has been entered. No matter what number you give to the first line of your program, the first code is always stored at 17129. Our program lets you look at the first ten code numbers.

For the moment, ignore the first two numbers in the series, and concentrate on the third and fourth, which are 10 and 0. These constitute the line number. Two codes are used, because we can use line numbers up to 65529, which uses two bytes. The line number is stored in two parts: The lower part comes first, and the upper part comes second. The actual decimal number is determined by adding the lower part to 256 multiplied by the upper part—in this case, it's 10 + 256*0. Line 1000 would be 232,3, because 232 + 3*256 = 1000. Table 1 shows how to convert decimal numbers to numbers in TRS-80 coded form.

The first and second numbers are the address number of the *next* line, so that the TRS-80 can pick up its directions at the start of each line. The numbers in our example should be 9,67, because the next line should start at address number 9 + 67*256 = 17161. When you PEEK at a piece of completely unused memory, you will find that the codes are alternately 0 and 255, which are the numbers that are set into the memory by the power-on sequence. Some bytes (low in memory address number) are set to other values when the BREAK key is pressed.

Whenever you type a line number, you use up five bytes of memory, consisting of a zero which is placed at the end of the previous line (to indicate that it has ended), the two bytes of the next line address, and the two bytes of the line number. Avoiding short lines wherever possible, even if it means using a lot of ELSE-IF statements, saves considerable amounts of memory, sometimes making the difference between being able to fit the program and getting the OM error signal.

Code number five on our list is 129. That's the TRS-80 code meaning FOR. You might expect this to be stored as three codes, the ASCII codes for F, O, and R, but the TRS-80 uses memory-saving single codes for all its commands. Table 1 shows a list of the command codes. After the FOR code, there's a blank (ASCII 32) because you typed in a blank between FOR and N (didn't you?). The computer does not need this space, and you can save memory by omitting all such spaces. There are only a few statements which can cause trouble if you do this to them—check the examples in the manual.

N appears as the seventh item in our list, stored as its ASCII equivalent, 78. The eighth item is the code for =, which is 213. This is *not* the same as

the ASCII code for =, because we're not using = as a character to be printed but as a command to be carried out. Since you left a gap between = and 17129, the ninth code number is 32.

The detective work begins to look interesting, and we would like to look beyond the tenth code. Program Listing 2 was a rather wasteful printing method, and we'll get more information on to the screen by using Program Listing 3, which prints the codes in lines separated by commas. We can now see the whole program in its coded form.

We recognize the first nine codes from the previous examples. The numbers 17129 and 17179 are stored in ASCII form, needing five bytes each. This is a wasteful method; if you use a number such as 17129 frequently in a program, you can save memory by declaring it as an integer variable. Use a command such as A% = 17129. The % sign means that the number is an integer and can be stored in two bytes using the code method we have seen used for line numbers. Alternatively, by using DEFINT A at the start of the program, we could command A = 17129 and achieve the same result. Each time we need 17129 in the program, we can now use A, saving memory space. The TO part of the FOR-TO-NEXT loop is stored, as usual, as a single-code number. Conversion of certain words, like FOR, NEXT, RUN, and so on, into single-byte numbers means that you have to be careful *not* to use these words as variable names in a program.

By checking the code numbers in Table 1, and the ASCII codes, you can trace how the instructions are coded—but this is just coded BASIC, not true machine code. The code numbers in a BASIC line are instructions to the interpreter. This introduces you fairly painlessly to the idea of instructions stored as number codes, and it shows beautifully how the TRS-80 line is coded. You can now see how it's possible to change line numbers. The first line number will always be found at memory locations 17131 and 17132. The bytes in 17129 and 17130 will indicate the address of the start of the next line; the third and fourth along from that number will give the next line number and so on.

Program Listing 4 searches through 16K memory and stops when it finds the address of line 5000. 5000 in TRS-80 code is 136,19, so we set the IF. . . statement to detect the sequence 136,19 anywhere in the memory. If you have a reference to this number as a variable earlier than line 5000, you'll turn up the wrong address, but it's simple to modify the program so that it lists every reference to 5000 (by adding a :NEXT at the end of line 10).

POKE

POKE is the companion command to PEEK, and it has to be followed by two numbers. The first number following POKE is the address which is to be used. The second number, separated from the first by a comma, is the

Dec. Code	BASIC Keyword	Dec. Code	BASIC Keyword
129	FOR	167	LOAD
130	RESET	168	MERGE
131	SET	169	NAME
132	CLS	170	KILL
133	CMD	171	RSET
134	RANDOM	172	RSET
135	NEXT	173	SAVE
136	DATA	174	SYSTEM
137	INPUT	175	LPRINT
138	DIM	176	DEF
139	READ	177	POKE
140	LET	178	PRINT
141	GOTO	179	CONT
142	RUN	180	LIST
143	IF	181	LLIST
144	RESTORE	182	DELETE
145	GOSUB	183	AUTO
146	RETURN	184	CLEAR
147	REM	185	CLOAD
148	STOP	186	CSAVE
149	ELSE	187	NEW
150	TRON	188	TAB
151	TROFF	189	TO
152	DEFSTR	190	FN
153	DEFINT	191	USING
154	DEFSNG	192	VARPTR
155	DEFDBL	193	USR
156	LINE	194	ERL
157	EDIT	195	ERR
158	ERROR	196	STRING$
159	RESUME	197	INSTR
160	OUT	198	POINT
161	ON	199	TIME$
162	OPEN	200	MEM
163	FIELD	201	INKEY$
164	GET	202	THEN
165	PUT	203	NOT
166	CLOSE	204	STP
205	+	231	CVS
206	–	232	CVD
207	*	233	EOF
208	/	234	LOC
209	↑	235	LOF
210	AND	236	MKI$
211	OR	237	MKS$
212	>	238	MKD$
213	=	239	CINT
214	<	240	CSNG

215	SGN	241	CDBL
216	INT	242	FIX
217	ABS	243	LEN
218	FRE	244	STR$
219	INP	245	VAL
220	POS	246	ASC
221	SQR	247	CHR$
222	RND	248	LEFT$
223	LOG	249	RIGHT$
224	EXP	250	MID$
225	COS		
226	SIN		
227	TAN		
228	ATN		
229	PEEK		
230	CVI		

Divide the number by 256 and discard anything which follows the decimal point. What's left is the upper byte (which always comes second in the coding sequence).
Multiply the upper byte number by 256 and subtract the result from the original number. The answer is now the lower byte (which comes first in the code sequence).

EXAMPLE: Convert 23478 to TRS-80 code: 23478/256 = 91.710937. All we want is the 91 (upper byte): 91 × 256 = 23296. Subtract from 23478, and we get 182 (lower byte).
The number in TRS-80 coding is therefore 182 91.

Table 1. *Converting decimal numbers to numbers in TRS-80 coded form.*

number between 0 and 255 which is to be inserted into that address in memory. Unlike PEEK, POKE changes the code stored in memory (unless you have used identical numbers), so it's a command you have to be careful about. You can't change the ROM which operates BASIC, but you can change a lot of codes elsewhere and end up with some very strange effects. A lot of poorly planned POKE instructions will cause the computer to start up again, with the words MEMORY SIZE? appearing. Unless you use the reset button at the back, you will lose any program which you had in memory, including machine-code programs as well as BASIC.

Most machine-code programs are loaded from system tapes, and we covered the techniques for loading these in Part II (see Volume 1 of the *Encyclopedia*). It always irks me to have to use a system tape for a short program, so I've rewritten common routines as BASIC programs, by using POKE. If I POKE a code number to a place in memory, it's as surely there as if I had put it there from a system tape. Program Listing 5 demonstrates the form of the BASIC program needed—this reads 100 bytes from a data line and POKEs them one by one into memory locations 32667 upwards.

If the machine-language program is short (less than 255 bytes), it is more convenient to read the characters into a string as illustrated in the Level II

manual. Program Listing 6 shows an example of this for a printer routine. The complete machine-code program exists as the long string ZZ$, and the program is accessed by using the address of the string in memory. The advantage of using a string for this purpose is that it doesn't need any answer to the MEMORY SIZE question, so nothing goes wrong if you forget to reserve memory. The disadvantage is that the address of a string in memory is not fixed. If the memory starts to fill up, the strings will be shifted about, and the computer will keep track of each. If a string gets shifted between two of the VARPTR instructions, however, the memory will get completely scrambled. For short routines and programs which use very little memory, it's fine.

For either of these methods, you still have to obtain machine code in the form of data lines. There are two ways of doing this. The first is to print out a decimal dump of the machine code, using a short BASIC program like Program Listing 5, setting the memory address numbers to the first and last addresses of the machine-code program which you placed in protected memory. This works well if the program is only twenty bytes or so, but it becomes decidedly tedious for longer programs.

POKE Graphics

You may also use the POKE instruction to obtain quick graphics. The video display of the TRS-80 is memory mapped, which means that every part of the screen corresponds to a piece of data in the memory. Because of the memory mapping, we can create shapes on the screen through POKE instructions to video memory, which runs from 15360 to 16383. Address 15360 controls the top left-hand side of the screen. The addresses go in bundles of 64 per line on the screen for 16 lines to 16383, which is at the bottom right-hand side. At each memory address, you can POKE numbers which will light up a screen block, just as SET does. Figure 1 shows what numbers correspond to which cells. If you type POKE 15360,5, you will light up the cells shown in the example at that part of the screen. POKE graphics require practice, but they run a lot faster than SET or CHR$( ) routines, and they are very useful when animations are needed. Program Listing 6 introduces a new TRS-80 BASIC function (and you thought you knew it all!). VARPTR(variable) is a form of the PEEK command and is applied to a named variable. If you have a number variable such as N in a program, the command PRINT VARPTR(N) puts the number, which is the memory address at which the variable N is stored, on the screen. If you then enter PRINT PEEK(VARPTR(N)), you'll get one byte of the variable itself—the byte which is stored at the VARPTR(N) address number. Integer variables need two bytes, so you'll have to PEEK(VARPTR(N) + 1) as well, and find the value by using the well-worn formula: lower byte + 256 * upper byte.

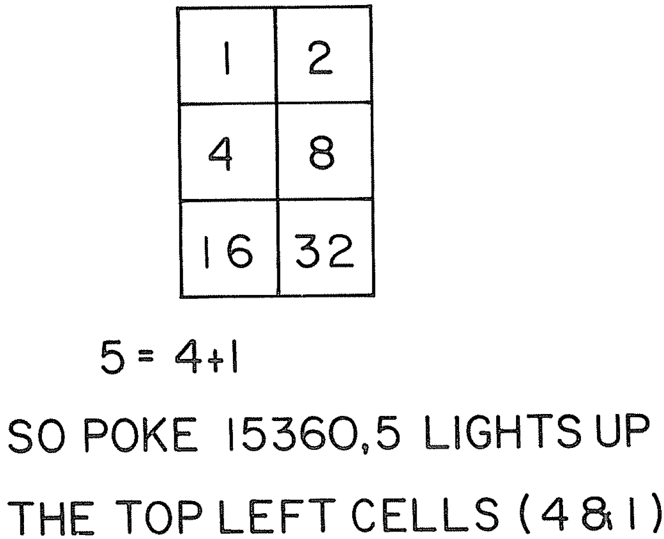


Figure 1

Single-precision numbers need four bytes of storage, and double-precision numbers need six bytes of storage, but the address of the first byte can always be obtained from the VARPTR command.

There's more information which you can get using the VARPTR command, but the address numbers are the most important. If you use the VARPTR command on a string variable, PEEK(VARPTR(string)) returns with the length of the string (a number of bytes not exceeding 255). The address of the string in memory is obtained from VARPTR(string) + 1, the lower byte, and VARPTR(string) + 2, which is the higher byte. This is the scheme used in Program Listing 6.

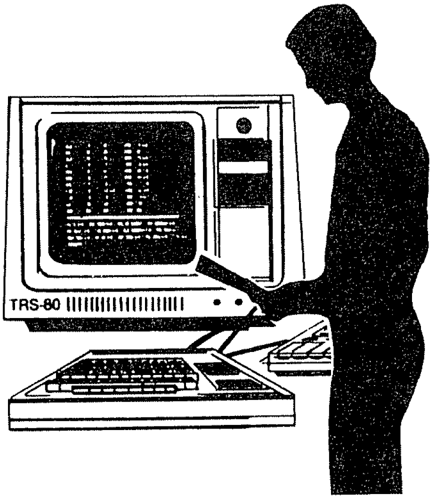
An important point about routines held in a string is that you must not erase the data or POKE instructions, because the TRS-80 resets all string variables each time you EDIT, CLEAR, or RUN. If the machine-code program is in high protected memory, then just switching off or giving a new answer to the MEMORY SIZE question will delete it.

USR(0) is used to insert a machine-code program in the middle of a BASIC program. For example, suppose you have the Radio Shack KBFIX program stored in high memory at address 32600, but the program has not been run. The problem is how to make the program run without having to go back to the SYSTEM command, type a slash, and enter the number 32600. The solution takes two steps. First, place the address of the start of the machine code into reserved memory at addresses 16526 and 16527. This lets the computer know where to find the machine-code program. Then, to go into the

machine-code program, use the command USR(0) with some other command, such as PRINT. The statement PRINT USR(0) would result in the machine-code program being run immediately after this statement was encountered. The PRINT part of the command is a dummy command, there only because the computer refuses to recognize USR(0) as an instruction by itself. Statements such as A = USR(0) are equally acceptable.

We're now approaching the end of "Into the 80s." I've purposely omitted quite a few instructions which you'll find useful later, simply because they're not particularly useful to you at this stage. The time has now come to sort out a few leftover items, and to give some advice on where to go from here, now that the keys of your TRS-80 look a little dull with wear.

By this time you've probably further explored the EDIT capabilities of the TRS-80. The manual is useful, with helpful examples, so you'll appreciate how powerful these EDIT subcommands are. Making full use of them can greatly reduce the time you spend programming, but you have to memorize the commands. Remember that whenever you use an EDIT command, you will lose all the values of variables. Don't type a lot of precious data into a program until you're sure that all the syntax errors are cleared, because the computer goes into edit mode automatically when a syntax error is detected. You can prevent this in various ways: Remember to press Q (for quit) whenever a snytax error appears, or use a line such as 2 ON ERROR GOTO 5000



and put STOP in line 5000 early in your program. If any error occurs, the program will jump to line 5000 and break without losing variables. You can then find the error code number by typing PRINT ERR/2 + 1 (the manual has a list of the error codes). The line number of the error can be found by typing PRINT ERL. These error-trapping routines can also be used inside

programs to help break out of errors which arise inevitably from the program, like reading too many data bytes—examples are given in the manual.

Where do you go from here? There's any amount of BASIC programming to do. Even if you run out of BASIC programs for your own use, there's a fair chance that there will be many people around you who need BASIC programs, but have no idea of how to write them. Lots of people earn a respectable living by writing programs, or by adapting existing programs, and as your skill improves you might find (as I do) that this is interesting and rewarding work, daylight or moonlight.

The other way you can go is into machine language, a much more difficult path. If, like me, you started computing in machine language before BASIC was invented, the path is easier, but for the complete beginner, the problem is to find a book which starts right at the beginning. With regret, I must report that the Radio Shack books which come with their Editor/Assembler package are not for the beginner, but the articles which appear in *80 Microcomputing* are a step in the right direction. I have also found a book called *Machine Language Programming from the Ground Up*, by Hubert S. Howe, Jr., which is excellent. Look out for it.

One final point. In this business you never stop learning—no one ever knows it all. No matter how long you have been using the TRS-80, you'll be able to thumb through *80 Microcomputing* some day and be struck dumb by some piece of information or some smart subroutine that had never struck you before. That's the best thing of all, because for me, while I'm learning, I'm living.

Program Listing 1

```
4 GOSUB INP E 80'S FIG 7.1
10 CLS:PRINT@154,CHR$(23)"TITLE":PRINTTAB(13)"=====":FO
     R N=1TO1500:NEXT:GOSUB200:REM  200 IS SUBROUTINE W
     HICH ASKS IS INSTRUCTIONS ARE NEEDED
20 CLS:PRINTTAB(35)"MENU":PRINT:PRINTTAB(2)"1. ENTER NE
     W DATA":PRINTTAB(2)"2. REPLAY DATA CASSETTE":PRINT
     TAB(2)"3. PROCESS DATA":PRINTTAB(2)"4. RECORD DATA
     ":PRINTTAB(2)"5. END PROGRAM"
30 GOSUB 500:REM INKEY$ ROUTINE TO FIND CHOICE
40 ON K GOSUB 1000,2000,3000,4000,5000
50 PRINT "DO YOU WANT TO RETURN TO THE MENU?":GOSUB 600
     :IF M=1 THEN 20 ELSE 5000:REM YES/NO RUTINE IS AT
     600
5000 END
```

Program Listing 2

```
10 FOR N= 17129 TO 17139:PRINT PEEK(N):NEXT
20 END:REM INTO THE 80'S FIG 7.2
```

Program Listing 3

```
10 FOR N=17129 TO 17179:PRINT PEEK(N);"    ";:NEXT
20 END:REM INTO THE 80'S FIG 7.5
```

Program Listing 4

```
5 REM NRO THE 80'S FIG 7.6
10 FOR N=17129 TO 32767: IF PEEK(N)<>136 AND PEEK(N+1)<>19 THEN
NEXT ELSE PRINT N
20 END
5000 PRINT"THIS IS LINE 5000"
```

Program Listing 5

```
10 FOR N=0 TO 99:READ J:POKE 32667+N,J:NEXT
20 DATA:REM NEED 100 NUMBERS BETWEEN 0 AND 255
```

Program Listing 6

```
1 CLEAR300:ZZ$="":FOR I=1TO107:READ J:ZZ$=ZZ$+CHR$(J):N
     EXT:POKE16422,PEEK(VARPTR(ZZ$)+1):POKE 16423,PEEK(
     VARPTR(ZZ$)+2)
5000 DATA255,243,121,254,13,40,3,254,32,216,245,229,197
     ,6,9,55,245,245,33,1,252,205,33,2,33,222,0,43,124,
     181,32,251,241,31,245,48,19,33,2,252,24,19,14,3,17
     5,13,40,2,24,219,0,0,0,0,24,47,198,0,33,1,252,205,
     33,2,0,0,33,222,0,43,124,181,32,251
5001 DATA16,212,17,222,0,203,74,40,11,33,2,252,205,33,2
     ,27,122,179,32,251,241,241,254,13,40,198,183,40,19
     7,193,225,241,201
```

UTILITY

Spool and Despool

Renumbering Made Easy

Spool and Despool

by H. S. Gentry

Simultaneous Peripheral Output OverLap (SPOOL) is a technique used by most large computer systems to prevent program delay because a slow peripheral, like a printer, is not ready. The output data is written (spooled) on a mass storage device and then transferred (despooled) when the peripheral is ready.

Spool

The TRS-80 spooler system is divided into two major sections, SPOOL and DESPOOL. The first of these sections is the output spooler, shown in Program Listing 1.

The code in line numbers 300 through 440 requests the file name and places it in the device control block (DCB) for the file. Line numbers 470 through 540 open an existing file or create a new one and check for errors. If any error is found, an error message is printed and the spool operation is terminated. If the file opens without error, then lines 550 through 590 connect the spooler to the printer DCB and return control to the operating system.

Now, each time the operating system (DOS, BASIC, etc.) attempts to print a character, the code in lines 650 through 930 is activated. The character is counted and stored in a 256-byte buffer. When this buffer is full, it is written to the disk. This procedure continues as long as the user allows it or until an error is detected.

When the spool operation is completed, you must close the spool file. This is necessary for two reasons. First, the data printed may not have ended on a 256-byte boundary. Thus, some data may be in the buffer that has not been written to the file. Closing SPOOL will detect this situation, set the unused area of the buffer to zeros, and write the last buffer to the file. The second reason is that the system program CLOSE must be called to update the disk directory.

The spool system performs both of these close operations, if control is transferred to label KLOSE (location FE76H in Program Listing 1). This may be done by entering DEBUG and typing GFE76. The memory containing the KLOSE program, the file DCB, the pointers, and the 256 byte buffer must not be changed until the close operation is done.

If you don't like using DEBUG to close your file you can create a close program as follows: load (but don't execute) the SPOOL program, then

dump the KLOSE part of SPOOL to a disk file called CLOSE/CMD. Don't dump more memory than needed. Actually, you only need an execution (transfer) address.

The dump command to close the file for the SPOOL in Program Listing 1 is: DUMP CLOSE/CMD:0 (START=X'FE76',END=X'FE9D',TRA=X'FE76'). Now, after your spool operation is finished, return to DOS and type CLOSE. The file is then closed and the spool operation terminated. You are left with an ASCII file containing all the printer output since the spool was started.

Despool

If you want to print a copy of the spool file the command PRINT could be used. However, this ties up the system while the printer is running. Fortunately, there is a better way, DSPOOL, shown in Program Listing 2. This program opens the spool file for printing and returns to the operating system. The data in the file is then printed while you perform almost any other job on your system. That's right, you can run a BASIC program or perform other disk operations while the file is being printed.

There are only a few exceptions: You cannot re-boot the system, you cannot write to the spool file while despooling, you cannot print data in the regular DOS manner until the despool is completed, you cannot spool one file while despooling another. The last restriction is included only because SPOOL and DSPOOL use the same memory.

If you move one of the programs to another location, you could SPOOL and DSPOOL at the same time, although you still may not write and read the same file at one time. You must use two different file names.

DSPOOL uses two links to the operating system, one to the 25-millisecond interrupt and another to the keyboard driver. The TRS-80 hardware interrupts the microcomputer forty times per second. The operating system uses this interrupt to run foreground tasks. These tasks include the real time clock, TRACE, or any job you'd like to run.

To run a given job you need to store the address of a pointer in the 25-millisecond queue list. The queue list is at memory location 4510H and 4511H. The pointer is two memory bytes containing the address of your program. This is a little confusing so let's look at Program Listing 2 to see what it means.

Lines 800 through 850 put the address of something called PINT in locations 4510H and 4511H. Notice that the code also saves the former contents of 4510H, 4511H to be put back later. PINT is a pointer that contains the memory address of your program.

In this example, 4510H, 4511H contains FD7A (the address of PINT) and FD7AH contains the address of INTHDL (FD7CH). Now, every 25 milliseconds INTHDL, the interrupt handler, is run.

INTHDL

The function of the DSPOOL interrupt handler INTHDL is very simple. It checks the RS-232 board to see if it will accept an output character. If the RS-232 board is not ready, INTHDL returns to the operating system. If a character can be output, INTHDL checks CCNT.

As long as CCNT is zero, INTHDL returns to the system. If it isn't, one character is output and counted. If the character is a carriage return, the buffer is set up to output a line feed. As long as there is data in the buffer, INTHDL will print it. All of this takes place in time stolen from your other work by the interrupt.

Getting data to the buffer is SCAN's job. SCAN reads one record every time the print buffer is empty (CCNT = 0). It is linked to the TRS-80 keyboard driver and runs every time the system checks the keyboard for input. If there is data in the buffer, SCAN returns control to the keyboard driver. But, if the buffer is empty, SCAN performs a file read, delaying the keyboard input for about one second.

If all the data has been read from the file, SCAN disconnects the DSPOOL program. If your printer is 110 baud, the disk reads occur about every 30 seconds. The spool system does not drive any printer faster than 40 characters per second (one per interrupt).

If your printer is faster than this, it will slow down to 40 cps. At 40 cps the disk reads occur about every 7.5 seconds. If reading at this rate interferes with the keyboard too much, then add a counter to INTHDL to slow the printer and thus the reads.

Another technique that reduces disk reads is reading two (or more) sectors at a time. However, this complicates the procedure used to find the end of the data.

Modifications

The DSPOOL program shown in Program Listing 2 is for a serial printer using the Radio Shack RS-232 board. The program can be used with a parallel printer (such as the standard printers sold by Radio Shack) by making a few changes.

Delete lines 370 through 500 and move the label SETUP to line 510. Replace lines 1210 through 1230 with the code in Program Listing 3. Replace line 1350 with LD(37E8H),A.

If your printer automatically feeds a line on every carriage return then delete lines 1360 through 1370 and lines 1430 through 1500.

If you use SPOOL-DSPOOL with NEWDOS or NEWDOS 80, it works as is. If you use it with TRSDOS 2.1, TRSDOS 2.2, or VTOS 3.0, you must add DEC HL between lines 860 and 870. This is necessary because the NEWDOS DCB maintains the number of sectors in a file, while the other systems maintain the number of sectors plus one.

If you use TRSDOS 2.2, change the program ORG and move both programs down to allow at least 51 unused bytes at the top of memory. Remember the end of the program is not the end of the memory it uses. Both SPOOL and DSPOOL use 256 bytes of memory starting at BUFFER. If BUFFER is at FE69H, the program uses memory up to FF69H.

It is also necessary to change the program ORG if you have less than 48K of memory or if a program is already using the top of your memory. Another useful modification replaces the 32 blanks in INBFR (line 2160 in DSPOOL, line 1210 in SPOOL) with a file name. For example: INBFR DEFW 'PRINTFIL/LST '. (Be sure to include enough spaces after the file name and before the last quote mark to make a total of 32 characters.) Then delete the code that requests the file specification (lines 500 through 680 in DSPOOL, lines 300 through 450 in SPOOL). The system then uses 'PRINTFIL/LST' as the SPOOL, DSPOOL file and you don't need to answer the filespec question.

Operation

Operating the SPOOL-DSPOOL system is very easy. Assemble the programs and create the disk files using NEWDOS EDTASM, the Radio Shack EDTASM and TAPEDISK or any other assembler. I use SPOOL/CMD as the file name for the spooler and DSPOOL/CMD for the despooler.

To use the system you need only type SPOOL when you want the spooling to begin and answer the FILESPEC? question with the name of the file that is to hold the printer output. If you want to spool BASIC output, you must run SPOOL before you go to BASIC, unless you have NEWDOS.

With NEWDOS you can run the SPOOL-DSPOOL system from BASIC with the CMD"XXX" command. When all of your printer output is spooled return to DOS and type "CLOSE" (or type CMD"CLOSE" from NEWDOS BASIC). When you are ready to print the file type "DSPOOL" and answer the FILESPEC? question with the same filespec used to spool the output. When the system returns to DOS, you may run another job, as long as you follow the rules.

While DSPOOL is running, the character in the lower right corner of the TRS-80 video display will flash. This indicates that DSPOOL is running. If you do not like this feature, delete lines 1180 through 1200 in DSPOOL.

Summary

The source code given in the listings is for the NEWDOS Editor-Assembler. You can easily change the code for any other assembler. Don't forget the rules given above. Always close your spool file when you are finished, and be sure to protect the memory used by these programs when in BASIC. Don't attempt to use CLOSE to close the read file after you run DSPOOL. It's not necessary and won't work.

If you have two disk drives, you can use one entire diskette to spool printer output. If you have only one drive, your spooling is limited, but you should be able to accumulate several pages of output before you must DSPOOL. Either way SPOOL-DSPOOL should improve your TRS-80 throughput.

This program is designed to work with TRSDOS 2.1, 2.2, and 2.3, and with NEWDOS 2.1.

Program Listing 1. *SPOOL*

```
                ;THIS IS THE PRINTER SPOOLER - WHEN LOADED
                ;IT WILL INTERCEPT ALL PRINTER OUTPUT AND
                ;STORE IT IN A 256 BYTE BUFFER WHEN THE
                ;BUFFER IS FULL THE DATA IS WRITTEN TO
                ;THE SPECIFIED FILE.  THE SPOOL FILE MUST
                ;BE CLOSED BY RUNNING THE SYSTEM PROGRAM
                ;CLOSE.
                ;
4467            DISP    EQU     4467H
0040            INPUT   EQU     40H
4424            OPEN    EQU     4424H
4436            READ    EQU     4436H
4026            PRDD    EQU     4026H
402D            DOS     EQU     402DH
4428            CLOSE   EQU     4428H
4420            INIT    EQU     4420H
443C            WRITE   EQU     443CH
                ;
FE00                    ORG     0FE00H
FE00 21C5FE     SETUP   LD      HL,MSG1         ;LOG ON
FE03 CD6744             CALL    DISP
FE06 21A4FE             LD      HL,INBFR
FE09 0620               LD      B,32
FE0B CD4000             CALL    INPUT
FE0E 78                 LD      A,B             ;GET ACTUAL #
FE0F B7                 OR      A
FE10 28EE               JR      Z,SETUP         ;NO INPUT
FE12 EB                 EX      DE,HL
FE13 83                 ADD     A,E             ;ADDRESS+#
FE14 6F                 LD      L,A             ;LOW ADDRESS
FE15 7A                 LD      A,D             ;HI ADD
FE16 CE00               ADC     A,0
FE18 67                 LD      H,A             ;HI ADDRESS
FE19 3620               LD      (HL),20H        ;BLANK CR
                ;INBFR NOW HAS FILE SPEC WITH TRAILING BLANKS
                ;INIT THE FILE
FE1B 21E1FE             LD      HL,BUFFER       ;PLACE
FE1E 11A4FE             LD      DE,INBFR        ;DCB
FE21 0600               LD      B,0
FE23 CD2044             CALL    INIT            ;OPEN IT
FE26 2809               JR      Z,OK            ;Z=1 IF OK
FE28 21D5FE             LD      HL,ERM
FE2B CD6744             CALL    DISP
FE2E C32D40             JP      DOS             ;AND GET OUT
FE31 2A2640     OK      LD      HL,(PRDD)       ;OLD DRIVER
FE34 22A2FE             LD      (SAVDD),HL      ;SAVE IT
FE37 2140FE             LD      HL,DRIVE        ;NEW DEIVER
FE3A 222640             LD      (PRDD),HL       ;PUT IT IN
FE3D C32D40             JP      DOS             ;DONE
                ;FILE IS OPEN - THIS IS THE ACTUAL DRIVER
                ;IT WILL STUFF THE CHARACTERS IN THE BUFFER
                ;IF THE BUFFER IS FULL A WRITE TO THE DISK
                ;WILL BE DONE.
                ;
FE40 E5         DRIVE   PUSH    HL
FE41 F5                 PUSH    AF
FE42 2A9EFE             LD      HL,(PRT)        ;POINT TO BUFFER
FE45 71                 LD      (HL),C          ;SAVE CHARACTER
FE46 23                 INC     HL
FE47 229EFE             LD      (PRT),HL
FE4A 3AA0FE             LD      A,(CCNT)        ;COUNT
FE4D FEFF               CP      0FFH            ;DUN
FE4F 2807               JR      Z,OUT
FE51 3C                 INC     A               ;COUNT IT
FE52 32A0FE             LD      (CCNT),A        ;PUT IT BACK
FE55 F1         POP     POP     AF
FE56 E1                 POP     HL
FE57 C9                 RET                     ;GO BACK
```

```
FE58 C5         00790 OUT     PUSH    BC
FE59 D5         00800         PUSH    DE
FE5A DDE5       00810         PUSH    IX
FE5C FDE5       00820         PUSH    IY
FE5E 11A4FE     00830         LD      DE,INBFR         ;DCB
FE61 CD3C44     00840         CALL    WRITE
FE64 21E1FE     00850         LD      HL,BUFFER
FE67 229EFE     00860         LD      (PRT),HL         ;RESTORE POINTER
FE6A AF         00870         XOR     A                ;A=0
FE6B 32A0FE     00880         LD      (CCNT),A
FE6E FDE1       00890         POP     IY
FE70 DDE1       00900         POP     IX
FE72 D1         00910         POP     DE
FE73 C1         00920         POP     BC
FE74 18DF       00930         JR      POP
                00940 ;THIS IS THE CLOSE ROUTINE - CALLED BY
                00950 ;THE CLOSE FUNCTION TO CLOSE OUT THE LAST
                00960 ;RECORD AND THEN CLOSE THE FILE
FE76 3AA0FE     00970 KLOSE   LD      A,(CCNT)         ;COUNT
FE79 B7         00980         OR      A
FE7A 2813       00990         JR      Z,KLOS           ;NO DATA CLOSE FILE
                01000 ;DATA IN FILE - NULL REMMAINDER THEN WRITE AND CLOSE
FE7C 2A9EFE     01010         LD      HL,(PRT)
FE7F 3600       01020 LOPC    LD      (HL),0
FE81 FEFF       01030         CP      0FFH             ;DUN
FE83 2804       01040         JR      Z,WRIT           ;FULL WRIT IT
FE85 3C         01050         INC     A
FE86 23         01060         INC     HL
FE87 18F6       01070         JR      LOPC
                01080 ;THIS IS THE WRIT TO THE DISK ROUTINE
FE89 11A4FE     01090 WRIT    LD      DE,INBFR         ;DCB
FE8C CD3C44     01100         CALL    WRITE
                01110 ;THIS IS THE CLOSE ROUTINE - IT WILL CLOSE THE
                01120 ;FILE
FE8F 11A4FE     01130 KLOS    LD      DE,INBFR         ;DCB
FE92 CD2844     01140         CALL    CLOSE
FE95 2AA2FE     01150         LD      HL,(SAVDD)
FE98 222640     01160         LD      (PRDD),HL        ;RESTORE PRINTER
FE9B C32D40     01170         JP      DOS              ;DONE
FE9E E1FE       01180 PRT     DEFW    BUFFER
FEA0 0000       01190 CCNT    DEFW    0
FEA2 0000       01200 SAVDD   DEFW    0
FEA4 20         01210 INBFR   DEFM    '
FEC5 53         01220 MSG1    DEFM    'SPOOL FILESPEC?'
FED4 03         01230         DEFB    3
FED5 53         01240 ERM     DEFM    'SPOOL ERROR'
FEE0 03         01250         DEFB    3
FEE1 00         01260 BUFFER  DEFB    0
FE00            01270         END     SETUP
00000 TOTAL ERRORS
```

```
BUFFER FEE1 01260   00470 00850 01180
CCNT   FEA0 01190   00710 00750 00880 00970
CLOSE  4428 00240   01140
DISP   4467 00180   00310 00530
DOS    402D 00230   00540 00590 01170
DRIVE  FE40 00650   00570
ERM    FED5 01240   00520
INBFR  FEA4 01210   00320 00480 00830 01090 01130
INIT   4420 00250   00500
INPUT  0040 00190   00340
KLOS   FE8F 01130   00990
KLOSE  FE76 00970
LOPC   FE7F 01020   01070
MSG1   FEC5 01220   00300
OK     FE31 00550   00510
OPEN   4424 00200
OUT    FE58 00790   00730
POP    FE55 00760   00930
PRDD   4026 00220   00550 00580 01160
PRT    FE9E 01180   00670 00700 00860 01010
READ   4436 00210
SAVDD  FEA2 01200   00560 01150
SETUP  FE00 00300   00370 01270
WRIT   FE89 01090   01040
WRITE  443C 00270   00840 01100
```

Program Listing 2. *DSPOOL*

```
                00100 ;DSPOOL  - 09 OCT. 1979 - H. S. GENTRY
                00110 ;PRINTER DE-SPOOLER - WHEN LOADED IT CONNECTS
                00120 ;TO THE 25MS INTERRUPT AND TO THE KEYBOARD
                00130 ;SCAN ROUTINE. THE SPECIFIED FILE WILL BE
                00140 ;LOADED ONE RECORD AT A TIME INTO LOCAL BUFFER
                00150 ;AND THE INTERRUPT HANDLER WILL PRINT ONE
                00160 ;CHARACTER EACH TIME THE PRINTER IS READY.
                00170 ;WHEN THE EOF IS FOUND THE LINK TO THE
                00180 ;INTERRUPT HANDLER AND THE KEYBOARD SCAN
                00190 ;IS REMOVED.
                00200 ;
4467            00210 DISP    EQU     4467H           ;DISPLAY MESSAGE
0040            00220 INPUT   EQU     40H             ;INPUT MESSAGE
4424            00230 OPEN    EQU     4424H           ;OPEN A FILE
4436            00240 READ    EQU     4436H           ;READ A FILE
4510            00250 MS25    EQU     4510H           ;25 MS QUEUE
4016            00260 KBDD    EQU     4016H           ;POINTER TO KEYBOARD
402D            00280 DOS     EQU     402DH           ;RTN TO DOS
00EA            00290 CNTREG  EQU     0EAH            ;CONTROL/STAT UART
00EB            00300 DTAREG  EQU     0EBH            ;DATA
3FFF            00320 ALIV    EQU     3FFFH
00E8            00330 RESURT  EQU     0E8H
00E9            00340 SWITCH  EQU     0E9H
                00350 ;
FD00            00360         ORG     0FD00H
FD00 211DFE     00510 SETUP   LD      HL,MSG1
FD03 CD6744     00520         CALL    DISP
FD06 21F6FD     00530         LD      HL,INBFR
FD09 0620       00540         LD      B,32
FD0B CD4000     00550         CALL    INPUT
FD0E 78         00560         LD      A,B             ;GET ACTUAL #
FD0F B7         00570         OR      A
FD10 28EE       00580         JR      Z,SETUP         ;NO INPUT
FD12 EB         00590         EX      DE,HL
FD13 83         00600         ADD     A,E             ;ADDRESS+#
FD14 6F         00610         LD      L,A             ;LOW ADDRESS
FD15 7A         00620         LD      A,D             ;HI ADD
FD16 CE00       00630         ADC     A,0
FD18 67         00640         LD      H,A             ;HI ADDRESS
FD19 3620       00650         LD      (HL),20H        ;BLANK CR
                00660 ;INBFR NOW HAS FILE SPEC WITH TRAILING BLANKS
                00670 ;INTERRUPT DRIVER IS LINKED ANY TIME CCNT IS
                00680 ;NOT ZERO IT WILL PUT OUT THE NEXT CHARACTER
                00690 ;
                00700 ;NOW TIME TO OPEN THE SPOOL FILE
FD1B 213DFE     00710         LD      HL,BUFFER       ;PLACE TO PUT DATA
FD1E 11F6FD     00720         LD      DE,INBFR        ;DCB
FD21 0600       00730         LD      B,0             ;LRL=0
FD23 CD2444     00740         CALL    OPEN
FD26 2809       00750         JR      Z,OK            ;Z=1 IF OK
FD28 212EFE     00760         LD      HL,ERM
FD2B CD6744     00770         CALL    DISP
FD2E C32D40     00780         JP      DOS             ;AND GET OUT
                00790 ;LINK 25 MS DRIVER
FD31 F3         00800 OK      DI
FD32 2A1045     00810         LD      HL,(MS25)       ;OLD ONE
FD35 223BFE     00820         LD      (SAV25),HL      ;SAVE IT
FD38 2162FD     00830         LD      HL,PINT         ;POINTER
FD3B 221045     00840         LD      (MS25),HL       ;LINK
FD3E FB         00850         EI
FD3F 2A002FE    00860         LD      HL,(SEC)        ;GET SECTORS
FD42 22F3FD     00870         LD      (SECTOR),HL
FD45 3AFEFD     00880         LD      A,(BX)          ;GET BYTES TO EOF
FD48 32F5FD     00890         LD      (BCNT),A
                00900 ;FILE OPEN OK NOW LINK KBD SCAN AND GET OUT
                00910 ;KBD SCAN WILL THEN FIND BUFFER EMPTY
                00920 ;AND READ A RECORD.
```

```
FD4B 2A1640   00930          LD      HL,(KBDD)       ;GET OLD ADDRESS
FD4E 22A5FD   00940          LD      (KEY),HL        ;SAVE FOR CONTINUE
FD51 2195FD   00950          LD      HL,SCAN         ;NEW SCAN
FD54 221640   00960          LD      (KBDD),HL       ;LINKED
              00970 ;SCAN IS NOW LINKED. NEED ONLY TO ENABLE
              00980 ;INTERRUPTS AND GET BACK TO DOS.  SCAN WILL
              00990 ;BE RUN EVERY TIME KEYBOARD IS CHECKED
              01000 ;INTHDL WILL BE RUN EVERY 25 MS
FD57 C32D40   01010          JP      DOS             ;GET OUT
              01020 ;THIS IS THE BAUDE RATE TABLE
FD5A 22       01030 BDTABL   DEFB    22H
FD5B 44       01040          DEFB    44H
FD5C 55       01050          DEFB    55H
FD5D 66       01060          DEFB    66H
FD5E 77       01070          DEFB    77H
FD5F AA       01080          DEFB    0AAH
FD60 CC       01090          DEFB    0CCH
FD61 EE       01100          DEFB    0EEH
              01110 ;
              01120 ;THIS IS INTHDL THE INTERRUPT HANDLER
              01130 ;IT WILL PRINT A CHARACTER IF CCNT IS NOT
              01140 ;ZERO AND THE PRINTER IS READY.
FD62 64FD     01150 PINT     DEFW    INTHDL          ;POINTER TO INTHDL
FD64 F5       01160 INTHDL   PUSH    AF              ;SAVE AF
FD65 E5       01170          PUSH    HL
FD66 3AFF3F   01180          LD      A,(ALIV)
FD69 3C       01190          INC     A
FD6A 32FF3F   01200          LD      (ALIV),A
FD6D 3AE837   01210          LD      A,(37E8H)
FD70 E6F0     01215          AND     0F0H
FD72 FE30     01220          CP      30H
FD74 201C     01225          JR      NZ,CONT
FD76 2A1BFE   01240          LD      HL,(CCNT)       ;CHAR COUNT
FD79 7D       01250          LD      A,L
FD7A FE00     01260          CP      0
FD7C 2005     01270          JR      NZ,OTPT         ;PUT IT OUT
FD7E 7C       01280          LD      A,H             ;L=0 CHECK H
FD7F FE00     01290          CP      0
FD81 280F     01300          JR      Z,CONT          ;ALL ZERO GET OUT
FD83 2B       01310 OTPT     DEC     HL              ;-1
FD84 221BFE   01320          LD      (CCNT),HL       ;PUT IT BACK
FD87 2A19FE   01330          LD      HL,(ADDR)       ;GET ADDRESS OF CHAR
FD8A 7E       01340          LD      A,(HL)          ;DATA
FD8B 32E837   01350          LD      (37E8H),A
FD8E 23       01380          INC     HL              ;BUMP ADDRESS
FD8F 2219FE   01390          LD      (ADDR),HL
FD92 E1       01400 CONT     POP     HL
FD93 F1       01410          POP     AF
FD94 C9       01420          RET                     ;DONE GET OUT
              01510 ;
              01520 ;THIS IS SCAN - IT IS LINKED TO KEYBOARD SCAN
              01530 ;AND WILL WATCH CCNT. IF CCNT IS ZERO THEN
              01540 ;SCAN WILL READ A RECORD. IF EOF IS FOUND OR
              01550 ;ANY READ ERROR IS ENCOUNTERED SCAN WILL
              01560 ;DISCONNECT ITSELF AND THE 25 MS HANDLER
              01570 ;
FD95 F5       01580 SCAN     PUSH    AF
FD96 E5       01590          PUSH    HL
FD97 2A1BFE   01600          LD      HL,(CCNT)
FD9A 7D       01610          LD      A,L
FD9B B7       01620          OR      A
FD9C 2004     01630          JR      NZ,EXIT
FD9E 7C       01640          LD      A,H
FD9F B7       01650          OR      A
FDA0 2805     01660          JR      Z,RRCD          ;YES READ RECORD
              01670 ;NOPE - RETURN TO KEYBOARD
FDA2 E1       01680 EXIT     POP     HL
FDA3 F1       01690          POP     AF
FDA4 C30000   01700          JP      0               ;DUMMY JUNP
FDA5          01710 KEY      EQU     $-2             ;BACK UP 2
```

Program continued

```
FDA7 C5         01720 RRCD     PUSH     BC
FDA8 D5         01730          PUSH     DE
FDA9 DDE5       01740          PUSH     IX
FDAB FDE5       01750          PUSH     IY
FDAD 11F6FD     01780          LD       DE,INBFR         ;DCB
FDB0 CD3644     01790          CALL     READ             ;READ RECORD
FDB3 2817       01800          JR       Z,OKR            ;READ OK SET COUND
                01810 ;NOT OK KILL EVERYTHING
FDB5 F3         01820 CLOS     DI                        ;STOP INTS.
FDB6 2A3BFE     01830          LD       HL,(SAV25)       ;OLD ADDRESS
FDB9 221045     01840          LD       (MS25),HL        ;PUT BACK
FDBC 2AA5FD     01850          LD       HL,(KEY)         ;OLD KBD
FDBF 221640     01860          LD       (KBDD),HL        ;PUT BACK
                01870 ;NOW POP REGISTERS AND RESTORE STACK
FDC2 FDE1       01880 POP      POP      IY
FDC4 DDE1       01890          POP      IX
FDC6 D1         01900          POP      DE
FDC7 C1         01910          POP      BC
FDC8 FB         01920          EI
FDC9 C3A2FD     01930          JP       EXIT
                01940 ;READ IT OK SET UP CCNT THEN GET OUT
FDCC 213DFE     01950 OKR      LD       HL,BUFFER
FDCF 2219FE     01960          LD       (ADDR),HL
FDD2 2AF3FD     01970          LD       HL,(SECTOR)      ;GET SECTORS
FDD5 7D         01980          LD       A,L              ;TEST
FDD6 FE00       01990          CP       0                ;ZERO?
FDD8 200D       02000          JR       NZ,DECIT         ;NOPE DEC IT AND STORE
FDDA 7C         02010          LD       A,H
FDDB FE00       02020          CP       0                ;HI =ZERO?
FDDD 2008       02030          JR       NZ,DECIT         ;NOPE
                02040 ;SECTOR COUNT=0, USE EOF BYTE COUNT NOT 256
FDDF 3AF5FD     02050          LD       A,(BCNT)
FDE2 6F         02051          LD       L,A
FDE3 2600       02060          LD       H,0
FDE5 1807       02070          JR       SCNT
FDE7 2B         02080 DECIT    DEC      HL
FDE8 22F3FD     02090          LD       (SECTOR),HL
FDEB 210001     02100          LD       HL,256
FDEE 221BFE     02110 SCNT     LD       (CCNT),HL
FDF1 18CF       02120          JR       POP              ;RESTORE AND GET OUT
                02130 ;
FDF3 0000       02140 SECTOR   DEFW     0
FDF5 00         02150 BCNT     DEFB     0
FDF6 20         02160 INBFR    DEFM     '
FE02            02170 SEC      EQU      INBFR+12
FDFE            02180 BX       EQU      INBFR+8
FE19 3DFE       02190 ADDR     DEFW     BUFFER
FE1B 0000       02200 CCNT     DEFW     0
FE1D 44         02210 MSG1     DEFM     'DSPOOL FILESPEC?'
FE2D 03         02220          DEFB     3
FE2E 44         02230 ERM      DEFM     'DSPOOL ERROR'
FE3A 03         02240          DEFB     3
FE3B 0000       02250 SAV25    DEFW     0
FE3D 00         02260 BUFFER   DEFB     0
FD00            02270          END      SETUP
00000 TOTAL ERRORS


ADDR    FE19 02190   01330 01390 01960
ALIV    3FFF 00320   01180 01200
BCNT    FDF5 02150   00890 02050
BDTABL  FD5A 01030
BUFFER  FE3D 02260   00710 01950 02190
BX      FDFE 02180   00880
CCNT    FE1B 02200   01240 01320 01600 02110
CLOS    FDB5 01820
CNTREG  00EA 00290
CONT    FD92 01400   01225 01300
DECIT   FDE7 02080   02000 02030
```

```
DISP    4467 00210   00520 00770
DOS     402D 00280   00780 01010
DTAREG  00EB 00300
ERM     FE2E 02230   00760
EXIT    FDA2 01680   01630 01930
INBFR   FDF6 02160   00530 00720 01780 02170 02180
INPUT   0040 00220   00550
INTHDL  FD64 01160   01150
KBDD    4016 00260   00930 00960 01860
KEY     FDA5 01710   00940 01850
MS25    4510 00250   00810 00840 01840
MSG1    FE1D 02210   00510
OK      FD31 00800   00750
OKR     FDCC 01950   01800
OPEN    4424 00230   00740
OTPT    FD83 01310   01270
PINT    FD62 01150   00830
POP     FDC2 01880   02120
READ    4436 00240   01790
RESURT  00E8 00330
RRCD    FDA7 01720   01660
SAV25   FE3B 02250   00820 01830
SCAN    FD95 01580   00950
SCNT    FDEE 02110   02070
SEC     FE02 02170   00860
SECTOR  FDF3 02140   00870 01970 02090
SETUP   FD00 00510   00580 02270
SWITCH  00E9 00340
```

Program Listing 3

```
01210     LD      A,(37E8H)
01215     AND     0F0H
01220     CP      30H
01225     JR      NZ,CONT
```

Renumbering Made Easy

by John Stratigakis

There comes a time in a person's life when it becomes necessary to renumber BASIC program lines. My time came when I had to add a line between lines 7 and 8 of a 150-line program. I solved this problem with a few PEEKs and POKEs. What could be simpler? A renumbering program, that's what! I developed a strong craving for a program that would change all line numbers and line references within a BASIC program. Then, one day, I was wandering through my local Radio Shack store, and I spotted a program called Renumber. Not wanting to part with my only ten-dollar bill, I made up my mind that if Tandy could do it, why couldn't I? I started to work on a quick and easy renumbering program.

First, a little knowledge of BASIC is necessary. Unlike Level I, Level II BASIC stores each command word as a one-byte code (see Table 1). This is much faster than storing each letter of the word, because the BASIC interpreter only has to check for one byte rather than four or five. Thus,

```
PRINT X/5
```

is stored as 5 bytes: B2 20 58 D0 35. This method of storage makes it easy to search for references (GOTO, GOSUB, etc.).

In addition, each line contains a zero at the end of the line, the line number in the form of two hex bytes, and a two-byte pointer, which is the address of the beginning of the next line. However, the pointer in the last program line does not point to the next line, but instead, it points to the end of the program—a pair of zero bytes. Therefore, the program

```
10 FOR X = 1 TO 1000:NEXT
```

is stored as follows (keep in mind that 42E9 is the beginning of BASIC program storage):

```
42E9                                                                     42FC
FC  42  0A  00  81  20  58  D5  31  20  BD  20  31  30  30  30  3A  87  00  00  00
        10      FOR     X   =   1       TO      1   0   0   0   :   NEXT
```

Notice that there are three zero bytes at the end: one for the end of the line, and two for the end of the program.

However, there is one matter that complicates things. Line references are not stored in hex. Rather, they are stored as ASCII numbers. Thus, GOTO 10 is stored as 8D 20 31 30 instead of 8D 20 0A 00.

The Program

First, I had to set the guidelines for the program (see Figure 1). This was the flowchart I used for Renumber 1.0, 2.0, and 3.0. Versions 1.0 and 2.0 were both total flops, since I had to hand-assemble them and key in the machine code with T-BUG. Then, I got my Editor/Assembler program, and version

3.0 went along smoothly. When it finally assembled correctly, I ran a test. I renumbered the following program:

```
5 FOR X=1 TO 1000:NEXT
10 GOTO 5
```

After renumbering, it looked like this:

```
10 FOR X=1 TO 1000:NEXT
20 GOTO 20
```

What happened was that the renumbering program changed old line 5 to line 10. The GOTO 5 became GOTO 10. Then, old line 10 became line 20. It was here that the problem occurred. The renumbering program assumed that GOTO 10 (which used to be GOTO 5) was a reference to old line 10. It

END	80	OUT	A0	VARPTR	C0	EXP	E0
FOR	81	ON	A1	USR	C1	COS	E1
RESET	82	OPEN	A2	ERL	C2	SIN	E2
SET	83	FIELD	A3	ERR	C3	TAN	E3
CLS	84	GET	A4	STRING$	C4	ATN	E4
CMD	85	PUT	A5	INSTR	C5	PEEK	E5
RANDOM	86	CLOSE	A6	POINT	C6	CVI	E6
NEXT	87	LOAD	A7	TIME$	C7	CVS	E7
DATA	88	MERGE	A8	MEM	C8	CVD	E8
INPUT	89	NAME	A9	INKEY$	C9	EOF	E9
DIM	8A	KILL	AA	THEN	CA	LOC	EA
READ	8B	LSET	AB	NOT	CB	LOF	EB
LET	8C	RSET	AC	STEP	CC	MKI$	EC
GOTO	8D	SAVE	AD	+	CD	MKS$	ED
RUN	8E	SYSTEM	AE	−	CE	MKD$	EE
IF	8F	LPRINT	AF	*	CF	CINT	EF
RESTORE	90	DEF	B0	/	D0	CSNG	F0
GOSUB	91	POKE	B1	↑	D1	CDBL	F1
RETURN	92	PRINT	B2	AND	D2	FIX	F2
REM	93	CONT	B3	OR	D3	LEN	F3
STOP	94	LIST	B4	>	D4	STR$	F4
ELSE	3A 95	LLIST	B5	=	D5	VAL	F5
TRON	96	DELETE	B6	<	D6	ASC	F6
TROFF	97	AUTO	B7	SGN	D7	CHR$	F7
DEFSTR	98	CLEAR	B8	INT	D8	LEFT$	F8
DEFINT	99	CLOAD	B9	ABS	D9	RIGHT$	F9
DEFSNG	9A	CSAVE	BA	FRE	DA	MID$	FA
DEFDBL	9B	NEW	BB	INP	DB	'	3A 93 FB
LINE	9C	TAB(	BC	POS	DC		FC
EDIT	9D	TO	BD	SQR	DD	.	FD
ERROR	9E	FN	BE	RND	DE	!	FE
RESUME	9F	USING	BF	LOG	DF	ISA	FF

Table 1

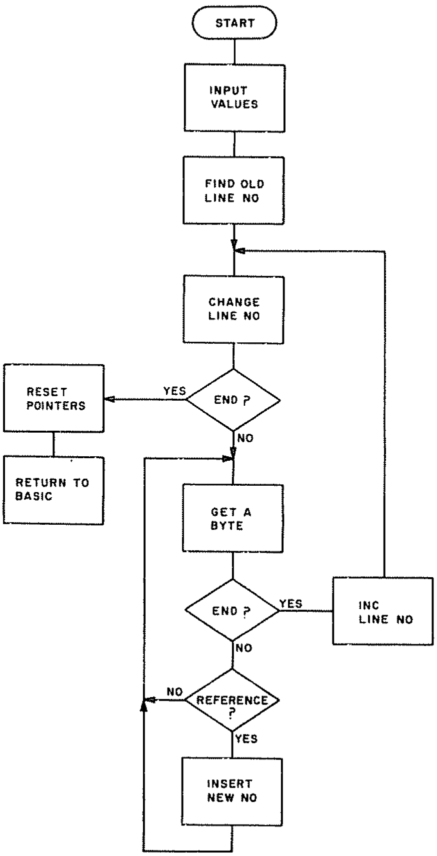


Figure 1

was therefore changed to GOTO 20, the line that old line 10 became. There was a simple cure to this problem. All I had to do was insert a byte before each changed line number to signal the renumbering program that "you already changed this one, dummy!" When renumbering was over, I would remove the markers.

Then came another disappointment. After renumbering a large program, I found that half of it had turned to garbage. I later learned that after one reference was changed, one of my pointers was no longer accurate. Because of this, my program was renumbering nonexistent line numbers and making other stupid mistakes. Not knowing what to do, I scrapped this version.

Now, I had to change the program logic. A look-up table seemed to be the answer. Using standard methods, though, to renumber a 16K program, I would need a 48K system just to hold the look-up table. After many sleepless

nights, I finally found the answer. Remember those two-byte line pointers? They would be a nice place to store old line numbers. All I would have to do is insert the old line number into the line pointer area of a line. Then, I would insert the new line number into the line number area of that line, and repeat the process for each line. (Notice that the line pointer and line number will be the same for a line that is not to be renumbered.) At this point, I would have a look-up table which shared memory with the resident BASIC program. Next, I would search the program for references. When I found one, I would search through the line pointers (old line numbers) until I found the one which matched the reference. Having done this, I would take the new line number from the line and insert it into the reference. When all references were done, I would reset all line pointers to make them accurate. (See Figure 2.) This was the basis for *working* version 4.0. 4.1 is merely a reworked 4.0 with a few minor changes.

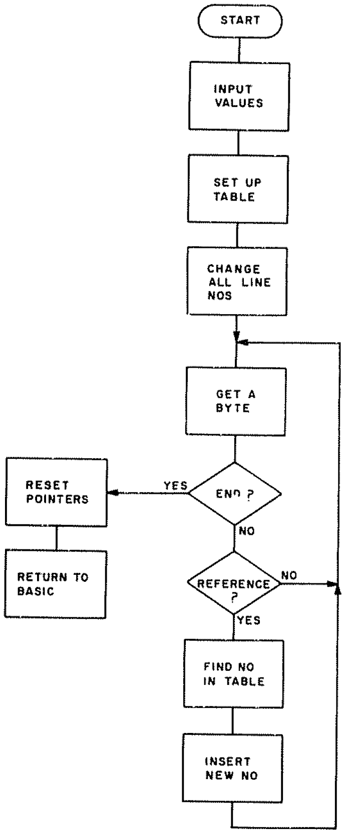


Figure 2

Using the Program

Renumber is located at 7D05-7F7A. I located it here so it would not interfere with KBFIX or Real-Time Clock. If this location conflicts with other programs you have, all you have to do is change the ORG statement in the source listing. However, if you do this, be sure to change the memory size and starting address of the program. Since I used 7D05-7F7A, I will use these addresses in describing the use of the program.

Upon power-up, answer MEMORY SIZE? with 32005 (decimal of 7D05). Type SYSTEM, and answer the prompt with the file name the program was saved under (I used RENUM). After the program is loaded, enter a /. RENUM is now operational. Should you happen to deactivate it, type SYSTEM and enter /32005, and it will again be activated.

To renumber a program, type NAME and press ENTER. The computer will respond with OLD START?. Answer this with the number of the first line to be renumbered (if you want to renumber the whole program, just press ENTER). Answer NEW START? with the line number to be used first (press ENTER for 10). Answer INCREMENT? with the increment to be used between line numbers (press ENTER for 10). When READY appears on the screen, the program will be renumbered with GOTO, GOSUB, THEN, ELSE, and RUN statements updated to reflect the new line numbers.

A few notes on the program:

1. *Never* renumber a program that has a line number of 0. Doing so will cause this version of RENUM to kill your program.
2. If you answer a question with a number that has five characters, you will not have to press ENTER.
3. Experiment on a program you have saved. This way you can find out the capabilities of RENUM.

Author's Note

The following changes are necessary so that my program will work on systems with an expansion interface:

1. Change line 190 from ORG 7D05H to ORG 7D00H
2. Delete line 4100
3. Change lines 250, 2120, and 3670 from HALT to JP 72H

The memory size is 32000, as is the start address.

Program Listing. *Renumber 4.1*

```
                 ;          RENUMBER 4.1
                 ;
                 ;WRITTEN BY JOHN STRATIGAKIS
                 ;
4280             OSTART  EQU     4280H
4288             NSTART  EQU     4288H
4290             INCREM  EQU     4290H
4200             BUFFER  EQU     4200H
42A0             BUFPTR  EQU     42A0H
7D05                     ORG     7D05H
                 ;SET "NAME" VECTOR
7D05 218E41      PROG    LD      HL,418EH
7D08 36C3                LD      (HL),0C3H
7D0A 21117D              LD      HL,RENUM
7D0D 228F41              LD      (418FH),HL
7D10 76                  HALT
                 ;INPUT VALUES
7D11 21547F      RENUM   LD      HL,OLD
7D14 CDA728              CALL    28A7H
7D17 CD067F              CALL    INPUT
7D1A AF                  XOR     A
7D1B BC                  CP      H
7D1C 2013                JR      NZ,OLD1
7D1E BD                  CP      L
7D1F 2010                JR      NZ,OLD1
                 ;DEFAULT
7D21 2AA440              LD      HL,(40A4H)
7D24 BE                  CP      (HL)
7D25 23                  INC     HL
7D26 2004                JR      NZ,SEARCH
7D28 BE                  CP      (HL)
7D29 CA4A1E              JP      Z,1E4AH ;FC
7D2C 23          SEARCH  INC     HL
7D2D 5E                  LD      E,(HL)
7D2E 23                  INC     HL
7D2F 56                  LD      D,(HL)
7D30 EB                  EX      DE,HL
7D31 228042      OLD1    LD      (OSTART),HL
7D34 21627F              LD      HL,NEW
7D37 CDA728              CALL    28A7H
7D3A CD067F              CALL    INPUT
7D3D AF                  XOR     A
7D3E BC                  CP      H
7D3F 2006                JR      NZ,NEW1
7D41 BD                  CP      L
7D42 2003                JR      NZ,NEW1
                 ;DEFAULT
7D44 210A00              LD      HL,0AH
7D47 228842      NEW1    LD      (NSTART),HL
7D4A 216E7F              LD      HL,INC
7D4D CDA728              CALL    28A7H
7D50 CD067F              CALL    INPUT
7D53 AF                  XOR     A
7D54 BC                  CP      H
7D55 2006                JR      NZ,INCRE
7D57 BD                  CP      L
7D58 2003                JR      NZ,INCRE
                 ;DEFAULT
7D5A 210A00              LD      HL,0AH
7D5D 229042      INCRE   LD      (INCREM),HL
                 ;SEARCH FOR OLD LINE NO.
7D60 2AA440              LD      HL,(40A4H)
7D63 1810                JR      TEST
7D65 23          TEST1   INC     HL
7D66 23          TEST2   INC     HL
7D67 3A8042              LD      A,(OSTART)
7D6A BE                  CP      (HL)
```

Program continued

```
7D6B 2813     00770          JR      Z,TEST3
7D6D 23       00780          INC     HL
7D6E 23       00790 TEST4    INC     HL
7D6F AF       00800          XOR     A
7D70 010000   00810          LD      BC,0
7D73 EDB1     00820          CPIR
7D75 AF       00830 TEST     XOR     A
7D76 BE       00840          CP      (HL)
7D77 20EC     00850          JR      NZ,TEST1
7D79 23       00860          INC     HL
7D7A BE       00870          CP      (HL)
7D7B 20E9     00880          JR      NZ,TEST2
7D7D C34A1E   00890          JP      1E4AH    ;FC ERROR
7D80 23       00900 TEST3    INC     HL
7D81 3A8142   00910          LD      A,(OSTART+1)
7D84 BE       00920          CP      (HL)
7D85 20E7     00930          JR      NZ,TEST4
7D87 2B       00940          DEC     HL
7D88 E5       00950          PUSH    HL
7D89 C1       00960          POP     BC
              00970 ;SET UP PART OF TABLE
7D8A 2AA440   00980          LD      HL,(40A4H)
7D8D E5       00990 NEXLN1   PUSH    HL
7D8E D1       01000          POP     DE
7D8F 23       01010          INC     HL
7D90 23       01020          INC     HL
7D91 E5       01030          PUSH    HL
7D92 B7       01040          OR      A
7D93 ED42     01050          SBC     HL,BC
7D95 E1       01060          POP     HL
7D96 2811     01070          JR      Z,NEXLIN
7D98 7E       01080          LD      A,(HL)
7D99 12       01090          LD      (DE),A
7D9A 23       01100          INC     HL
7D9B 13       01110          INC     DE
7D9C 7E       01120          LD      A,(HL)
7D9D 12       01130          LD      (DE),A
7D9E 23       01140          INC     HL
7D9F AF       01150          XOR     A
7DA0 C5       01160          PUSH    BC
7DA1 010000   01170          LD      BC,0
7DA4 EDB1     01180          CPIR
7DA6 C1       01190          POP     BC
7DA7 18E4     01200          JR      NEXLN1
              01210 ;SET UP REMAINDER OF TABLE
7DA9 7E       01220 NEXLIN   LD      A,(HL)
7DAA 12       01230          LD      (DE),A
7DAB 23       01240          INC     HL
7DAC 13       01250          INC     DE
7DAD 7E       01260          LD      A,(HL)
7DAE 12       01270          LD      (DE),A
7DAF 3A8942   01280          LD      A,(NSTART+1)
7DB2 77       01290          LD      (HL),A
7DB3 2B       01300          DEC     HL
7DB4 3A8842   01310          LD      A,(NSTART)
7DB7 77       01320          LD      (HL),A
7DB8 23       01330          INC     HL
7DB9 23       01340          INC     HL
7DBA AF       01350          XOR     A
7DBB 010000   01360          LD      BC,0
7DBE EDB1     01370          CPIR
7DC0 E5       01380          PUSH    HL
7DC1 D1       01390          POP     DE
7DC2 BE       01400          CP      (HL)
7DC3 23       01410          INC     HL
7DC4 2810     01420          JR      Z,NOLIN1
7DC6 23       01430 NOZER1   INC     HL
7DC7 E5       01440          PUSH    HL
7DC8 2A8842   01450          LD      HL,(NSTART)
7DCB ED4B9042 01460          LD      BC,(INCREM)
```

```
7DCF 09       01470          ADD    HL,BC
7DD0 228842   01480          LD     (NSTART),HL
7DD3 E1       01490          POP    HL
7DD4 18D3     01500          JR     NEXLIN
7DD6 BE       01510 NOLIN1   CP     (HL)
7DD7 20ED     01520          JR     NZ,NOZER1
              01530 ;SEARCH FOR LINE REFS.
7DD9 2AA440   01540          LD     HL,(40A4H)
7DDC 23       01550 INC1     INC    HL
7DDD 23       01560 INC2     INC    HL
7DDE 23       01570          INC    HL
7DDF 23       01580 GETBYT   INC    HL
7DE0 7E       01590          LD     A,(HL)
7DE1 FE22     01600          CP     22H
7DE3 200B     01610          JR     NZ,CHECK
7DE5 23       01620 QUOTES   INC    HL
7DE6 7E       01630          LD     A,(HL)
7DE7 FE22     01640          CP     22H
7DE9 28F4     01650          JR     Z,GETBYT
7DEB B7       01660          OR     A
7DEC 281E     01670          JR     Z,NEXT
7DEE 18F5     01680          JR     QUOTES
7DF0 FE8D     01690 CHECK    CP     8DH
7DF2 CA367E   01700          JP     Z,LINE
7DF5 FE8E     01710          CP     8EH
7DF7 CA367E   01720          JP     Z,LINE
7DFA FE91     01730          CP     91H
7DFC CA367E   01740          JP     Z,LINE
7DFF FE95     01750          CP     95H
7E01 CA367E   01760          JP     Z,LINE
7E04 FECA     01770          CP     0CAH
7E06 CA367E   01780          JP     Z,LINE
7E09 B7       01790          OR     A
7E0A 20D3     01800          JR     NZ,GETBYT
7E0C 23       01810 NEXT     INC    HL
7E0D AF       01820          XOR    A
7E0E BE       01830          CP     (HL)
7E0F 20CB     01840          JR     NZ,INC1
7E11 23       01850          INC    HL
7E12 BE       01860          CP     (HL)
7E13 20C8     01870          JR     NZ,INC2
              01880 ;RESTORE "NEXT LINE" POINTERS
7E15 2AA440   01890          LD     HL,(40A4H)
7E18 E5       01900 POINTR   PUSH   HL
7E19 D1       01910          POP    DE
7E1A AF       01920          XOR    A
7E1B BE       01930          CP     (HL)
7E1C 23       01940          INC    HL
7E1D 2003     01950          JR     NZ,NOTEND
7E1F BE       01960          CP     (HL)
7E20 280F     01970          JR     Z,END
7E22 23       01980 NOTEND   INC    HL
7E23 23       01990          INC    HL
7E24 23       02000          INC    HL
7E25 010000   02010          LD     BC,0
7E28 EDB1     02020          CPIR
7E2A 7D       02030          LD     A,L
7E2B 12       02040          LD     (DE),A
7E2C 13       02050          INC    DE
7E2D 7C       02060          LD     A,H
7E2E 12       02070          LD     (DE),A
7E2F 18E7     02080          JR     POINTR
7E31 23       02090 END      INC    HL
              02100 ;RESTORE "VARIABLE POINTER"
7E32 22F940   02110          LD     (40F9H),HL
7E35 76       02120          HALT
              02130 ;ROUTINE TO CHANGE LINE REF.
7E36 D7       02140 LINE     RST    10H
7E37 E5       02150          PUSH   HL
7E38 CD5A1E   02160          CALL   1E5AH
```

Program continued

```
7E3B C1         02170           POP     BC
7E3C E5         02180           PUSH    HL
7E3D C5         02190           PUSH    BC
                02200  ;SEARCH FOR OLD NO. IN TABLE
7E3E 2AA440     02210           LD      HL,(40A4H)
7E41 7E         02220  CHECK4   LD      A,(HL)
7E42 BB         02230           CP      E
7E43 23         02240           INC     HL
7E44 2816       02250           JR      Z,CHECK3
7E46 23         02260  CHECK5   INC     HL
7E47 23         02270           INC     HL
7E48 23         02280           INC     HL
7E49 AF         02290           XOR     A
7E4A 010000     02300           LD      BC,0
7E4D EDB1       02310           CPIR
7E4F BE         02320           CP      (HL)
7E50 20EF       02330           JR      NZ,CHECK4
7E52 23         02340           INC     HL
7E53 BE         02350           CP      (HL)
7E54 2B         02360           DEC     HL
7E55 20EA       02370           JR      NZ,CHECK4
7E57 C1         02380           POP     BC
7E58 E1         02390           POP     HL
7E59 C3E27E     02400           JP      COMMA
7E5C 7E         02410  CHECK3   LD      A,(HL)
7E5D BA         02420           CP      D
7E5E 20E6       02430           JR      NZ,CHECK5
                02440  ;FOUND
7E60 23         02450           INC     HL
                02460  ;FIND DESTINATION OF MOVE
7E61 E5         02470           PUSH    HL
7E62 210042     02480           LD      HL,BUFFER
7E65 22A042     02490           LD      (BUFPTR),HL
7E68 E1         02500           POP     HL
7E69 5E         02510           LD      E,(HL)
7E6A 23         02520           INC     HL
7E6B 56         02530           LD      D,(HL)
7E6C EB         02540           EX      DE,HL
7E6D 111027     02550           LD      DE,2710H
7E70 CD3B7F     02560           CALL    DIVIDE
7E73 11E803     02570           LD      DE,3E8H
7E76 CD3B7F     02580           CALL    DIVIDE
7E79 116400     02590           LD      DE,64H
7E7C CD3B7F     02600           CALL    DIVIDE
7E7F 110A00     02610           LD      DE,0AH
7E82 CD3B7F     02620           CALL    DIVIDE
7E85 110100     02630           LD      DE,1
7E88 CD3B7F     02640           CALL    DIVIDE
7E8B 2AA042     02650           LD      HL,(BUFPTR)
7E8E 3600       02660           LD      (HL),0
7E90 210042     02670           LD      HL,BUFFER
7E93 3E30       02680           LD      A,30H
7E95 BE         02690  CMPARE   CP      (HL)
7E96 2003       02700           JR      NZ,FOUND
7E98 23         02710           INC     HL
7E99 18FA       02720           JR      CMPARE
7E9B E5         02730  FOUND    PUSH    HL
7E9C D1         02740           POP     DE
7E9D 2AA042     02750           LD      HL,(BUFPTR)
7EA0 B7         02760           OR      A
7EA1 ED52       02770           SBC     HL,DE
7EA3 E5         02780           PUSH    HL
7EA4 C1         02790           POP     BC
7EA5 D1         02800           POP     DE
7EA6 D5         02810           PUSH    DE
7EA7 13         02820  DESTIN   INC     DE
7EA8 0B         02830           DEC     BC
7EA9 78         02840           LD      A,B
7EAA B1         02850           OR      C
7EAB 20FA       02860           JR      NZ,DESTIN
```

```
7EAD C1         02870           POP     BC
7EAE E1         02880           POP     HL
7EAF C5         02890           PUSH    BC
7EB0 E5         02900           PUSH    HL
                02910 ;FIND NO. OF BYTES TO MOVE
7EB1 010000     02920           LD      BC,0
7EB4 AF         02930           XOR     A
7EB5 2B         02940           DEC     HL
7EB6 23         02950 BYTES     INC     HL
7EB7 03         02960           INC     BC
7EB8 BE         02970           CP      (HL)
7EB9 20FB       02980           JR      NZ,BYTES
7EBB 23         02990           INC     HL
7EBC 03         03000           INC     BC
7EBD BE         03010           CP      (HL)
7EBE 20F6       03020           JR      NZ,BYTES
7EC0 23         03030           INC     HL
7EC1 03         03040           INC     BC
7EC2 BE         03050           CP      (HL)
7EC3 20F1       03060           JR      NZ,BYTES
7EC5 E1         03070           POP     HL
7EC6 D5         03080           PUSH    DE
                03090 ;MOVE!
7EC7 CDEC7E     03100           CALL    MOVE
7ECA E1         03110           POP     HL
7ECB D1         03120           POP     DE
7ECC E5         03130           PUSH    HL
                03140 ;INSERT ASCII FOR NEW REF.
7ECD 210042     03150           LD      HL,BUFFER
7ED0 3E30       03160           LD      A,30H
7ED2 BE         03170 CMPR1     CP      (HL)
7ED3 2003       03180           JR      NZ,LOAD
7ED5 23         03190           INC     HL
7ED6 18FA       03200           JR      CMPR1
7ED8 7E         03210 LOAD      LD      A,(HL)
7ED9 B7         03220           OR      A
7EDA 2805       03230           JR      Z,DONE
7EDC 12         03240           LD      (DE),A
7EDD 13         03250           INC     DE
7EDE 23         03260           INC     HL
7EDF 18F7       03270           JR      LOAD
7EE1 E1         03280 DONE      POP     HL
                03290 ;CHECK FOR COMMA
                03300 ;    (ON-GOTO,ON-GOSUB)
7EE2 2B         03310 COMMA     DEC     HL
7EE3 D7         03320           RST     10H
7EE4 FE2C       03330           CP      2CH
7EE6 CA367E     03340           JP      Z,LINE
7EE9 C3E17D     03350           JP      GETBYT+2
                03360 ;SUBROUTINE TO MOVE BYTES
7EEC E5         03370 MOVE      PUSH    HL
7EED D5         03380           PUSH    DE
7EEE C5         03390           PUSH    BC
7EEF E5         03400           PUSH    HL
7EF0 B7         03410           OR      A
7EF1 ED52       03420           SBC     HL,DE
7EF3 E1         03430           POP     HL
7EF4 3804       03440           JR      C,MOVE10
7EF6 EDB0       03450           LDIR
7EF8 1808       03460           JR      COMPLT
7EFA 09         03470 MOVE10    ADD     HL,BC
7EFB 2B         03480           DEC     HL
7EFC EB         03490           EX      DE,HL
7EFD 09         03500           ADD     HL,BC
7EFE 2B         03510           DEC     HL
7EFF EB         03520           EX      DE,HL
7F00 EDB8       03530           LDDR
7F02 C1         03540 COMPLT    POP     BC
7F03 D1         03550           POP     DE
7F04 E1         03560           POP     HL
```

Program continued

```
7F05 C9         03570          RET
                03580 ;SUBROUTINE TO INPUT VALUES
7F06 2AA740     03590 INPUT    LD      HL,(40A7H)
7F09 E5         03600          PUSH    HL
7F0A 0605       03610          LD      B,5
7F0C CD2B00     03620 KEYSCN   CALL    2BH
7F0F FE01       03630          CP      1
7F11 2006       03640          JR      NZ,NOBRK
7F13 3E0D       03650          LD      A,0DH
7F15 CD3300     03660          CALL    33H
7F18 76         03670          HALT
7F19 FE0D       03680 NOBRK    CP      0DH
7F1B 2811       03690          JR      Z,ENTER
7F1D FE30       03700          CP      30H
7F1F FA0C7F     03710          JP      M,KEYSCN
7F22 FE3A       03720          CP      3AH
7F24 F20C7F     03730          JP      P,KEYSCN
7F27 CD3300     03740          CALL    33H
7F2A 77         03750          LD      (HL),A
7F2B 23         03760          INC     HL
7F2C 10DE       03770          DJNZ    KEYSCN
7F2E 3600       03780 ENTER    LD      (HL),0
7F30 E1         03790          POP     HL
7F31 CD5A1E     03800          CALL    1E5AH
7F34 EB         03810          EX      DE,HL
7F35 3E0D       03820          LD      A,0DH
7F37 CD3300     03830          CALL    33H
7F3A C9         03840          RET
                03850 ;SUBROUTINE TO DIVIDE HL,DE
7F3B 0600       03860 DIVIDE   LD      B,0
7F3D B7         03870 LOOP10   OR      A
7F3E ED52       03880          SBC     HL,DE
7F40 3803       03890          JR      C,DONE10
7F42 04         03900          INC     B
7F43 18F8       03910          JR      LOOP10
7F45 19         03920 DONE10   ADD     HL,DE
7F46 E5         03930          PUSH    HL
7F47 2AA042     03940          LD      HL,(BUFPTR)
7F4A 78         03950          LD      A,B
7F4B C630       03960          ADD     A,30H
7F4D 77         03970          LD      (HL),A
7F4E 23         03980          INC     HL
7F4F 22A042     03990          LD      (BUFPTR),HL
7F52 E1         04000          POP     HL
7F53 C9         04010          RET
                04020 ;MESSAGES
7F54 0E0D       04030 OLD      DEFW    0D0EH
7F56 4F         04040          DEFM    'OLD START? '
7F61 00         04050          DEFB    0
7F62 4E         04060 NEW      DEFM    'NEW START? '
7F6D 00         04070          DEFB    0
7F6E 49         04080 INC      DEFM    'INCREMENT? '
7F79 00         04090          DEFB    0
7F7A 76         04100          HALT    ;PROTECTIVE END BLOCK
7D05            04110          END     PROG
00000 TOTAL ERRORS
```

APPENDIX

APPENDIX A

BASIC Program Listings

Debugging someone else's mistakes is no fun. In a business environment, where programs are continuously updated and programmers come and go, well-commented and structured programs are a must. Indeed, it behooves any serious programmer to learn structured technique.

The BASIC language has no inherent structure. Most interpreters allow remark lines and some are capable of ignoring unnecessary spacing, but BASIC is still more "Beginner's Instruction Code" than "All-purpose."

The listings in this encyclopedia are an attempt at formatting the TRS-80 BASICs. We think it makes them easier to read, easier to trace, and less imposing when it comes time to type them into the computer. You should *not*, however, type them in exactly as they appear. Follow normal syntax and entry procedures as described in your user's manual.

Level I Programs

Programs originally in Level I have been converted to allow running in Level II. To run in Level I, follow this procedure:

- Delete any dimension statements. Example: DIM A (25).
- Change PRINT@ to PRINTAT.
- Make sure that no INPUT variable is a STRING variable.
  Example: INPUT A$ would be changed to INPUT A and subsequent code made to agree.
- Abbreviate all BASIC statements as allowed by Level I.
  Example: *PRINT* is abbreviated *P*.

Model III Users

For the Model I, OUT255,0 and OUT255,4 turn the cassette motor off and on, respectively. For the Model III, change these statements to OUT236,0 and OUT236,2.

APPENDIX B

Glossary

A

access time—the elapsed time between a request for data and the appearance of valid data on the output pins of a memory chip. Usually 200-450 nanoseconds for TRS-80 RAM.

accumulator—traditionally the register where arithmetic (the accumulation of numbers) takes place.

acoustic coupler—a connection to a modem allowing signals to be transmitted through a regular telephone handset.

A/D converter—analog to digital converter. See D/A converter.

address—a code that specifies a register, memory location, or other data source or destination.

ALGOL—an acronym for ALGOrithmic Language. A very high-level language used in scientific applications.

algorithm—a predetermined process for the solution of a problem or completion of a task in a finite number of steps.

alignment—adjustment of hardware to achieve proper transfer of data. In the TRS-80 this usually applies to cassette heads and disk drives.

alphanumerics—refers to the letters of the alphabet and digits of the number system, specifically omitting the characters of punctuation and syntax.

ALU—Arithmetic-Logic Unit. Internal, and inaccessible to the programmer, it is the interface between registers and memory, manipulating them as necessary to perform the individual instructions of the microprocessor.

analog—data is represented electrically by varying voltages or amplitudes.

AND—a Boolean logical function. Two operators are tested and if both are true the answer is true. Truth is indicated by a high bit, or "1" in machine language, or a positive value in BASIC. If the operators are bytes or words, each element is tested separately.

APL—a popular high-level mathematical language.

argument—any of the independent variables accompanying a command.

ASCII—American Standard Code for Information Interchange. An almost universally accepted code (at least for punctuation and capital letters) where characters and printer commands are represented by numbers between 0 and 255 (base 10). The number is referred to as an ASCII code.

assembler—a piece of software that translates operational codes into their binary equivalents.

B

backup—refers to making copies of all software and data stored externally and having duplicate hardware available.

base—a mathematical term that refers to the number of digits in a number system. The decimal system, using digits 0 through 9, is called base 10. The binary system is base 2.

BASIC—an acronym for Beginner's All-purpose Symbolic Instruction Code. Developed at Dartmouth College and similar to FORTRAN. The standard, high-level, interactive language for microcomputers.

batch processing—a method of computing where many of the same type jobs or programs are done in one machine run. For example, a programming class may type programs on data cards and turn them over to the computer operator. All the cards are put into the card reader, and the results of each person's program are returned later. This is contrasted with interactive computing.

baud rate—a measure of the speed at which serial data is sent. The equivalent of bits per second (bps) in microcomputing.

benchmark—to test performance against a known standard.

binary—a number system which uses only 0 and 1 as digits. It is the equivalent of base 2. Used in microcomputing because it is easy to represent 1s and 0s by high and low electrical signals.

bit—an abbreviation for binary digit. A 0 or 1 in the binary number system. A single high or low signal in a computer.

Boolean algebra—a mathematical system of logic named after George

Boole. Routines are described by combinations of ANDs, ORs, XORs, NOTs, and IF-THENs. All computer functions are based upon these operators.

boot—short for bootstrap loader or the use of one. The bootstrap loader is a very short routine whose purpose is to load a more sophisticated system into the computer when it is first turned on. Sometimes it is keyed in, and on other machines it is in read only memory (ROM). Using this program is called "booting" the system or cold-starting.

bps—bits per second.

buffer—memory set aside temporarily for use by the program. Particularly refers to memory used to make up differences in the data transfer rates of the computer and external devices such as printers and disks.

bug—an error in software.

bus—an ordered collection of all address, data, timing, and status lines in the computer.

byte—eight bits that are read simultaneously as a single code.

C

CAI—an acronym for Computer Aided Instruction.

card—a specifically designed sheet of cardboard with holes punched in specific columns. The placement of the holes represents machine-readable data. Also a term referring to a printed circuit board.

carrier—a steady signal that can be slightly modified (modulated) continuously. These modulations can be interpreted as data. In microcomputers the technique is used primarily in modem communications and tape input/output (I/O).

character—a single symbol that is represented inside the computer by a specific code.

checksum—a method of detecting errors in a block of data by adding each piece of data in the block to a sum and comparing the final result to a predetermined result for that block of data.

chip—a physical package containing electrical circuits. They vary from aspirin-size for a simple timer to about the size of a stick of gum for a complete microprocessor.

clock—a simple circuit that generates the synchronization signals for the microprocessor. The speed or frequency of this clock directly affects the speed at which the computer can perform, regardless of the speed of which the individual chips are capable.

COBOL—COmmon Business Oriented Language. A language used primarily for data processing. Allows programming statements that are very similar to English sentences.

compiler—a piece of software that will convert a program written in a high-level language to binary code.

complement—a mathematical calculation. In computers it specifically refers to inverting a binary number. Any 1 is replaced by a 0, and vice versa.

concatenate—to put two things, each complete by itself, together to make a larger complete thing. In computers this refers to strings of characters or programs.

constant—a value that doesn't change.

CPU—Central Processing Unit. The circuitry that actually performs the functions of the instruction set.

CRT—Cathode Ray Tube. In computing this is just the screen the data appears on. A TV has a CRT.

cue—refers to positioning the tape on a cassette unit so that it is set up to read/write the right section of tape.

cycle—a specific period of time, marked in the computer by the clock.

D

D/A converter—digital to analog converter. Common in interfacing computers to the outside world.

data base—refers to a series of programs each having a different function

but all using the same data. The data is stored in one location or file and each program uses it in a fashion that still allows the other program to use it.

data entry—the practice of entering data into the computer or onto a storage device. Knowledge of operating or programming a computer is not necessary for a data entry operator.

debug—to remove bugs from a program.

decrement—to decrease the value of a number. In computers the number is in memory or a register, and the amount it is decremented is usually one.

dedicated—in computer terminology, a system set up to perform a single task.

default—that which is assumed if no specific information is given.

degauss—to demagnetize. Must be done periodically to tape and disk heads for reliable data transfer.

digital—all data is represented in binary code. In microcomputers, a high electrical signal is a 1 and a low signal is a 0.

disassembly—remaking an assembly source program from a machine-code program.

disk—an oxide-coated, circular, flat object, in a variety of sizes and containers, on which computer data can be stored.

disk controller—an interface between the computer and the disk drive.

disk drive—a piece of hardware that rotates the disk and performs data transfer to and from the disk.

disk operating system—(DOS) the system software that manipulates the data to be sent to the disk controller.

DMA—direct memory access. A process where the CPU is disabled or bypassed temporarily and memory is read or written to directly.

documentation—a collection of written instructions necessary to use a piece of hardware, software, or a system.

dot matrix printer—instead of each letter having a separate type head (like a typewriter), the single print head makes the characters by printing groups of dots. The print is not as easy to read, but such printers are less expensive to make.

driver—a small piece of system software used to control an external device such as a keyboard or printer.

dump—to write data from memory to an external storage device.

duplex—refers to two-way communications taking place independently, but simultaneously.

dynamic memory—circuits that require a periodic (every few milliseconds) recharge so that the stored data is not lost.

E

EAROM—an acronym for Electrically Alterable Read Only Memory. The chip can be read at normal speed, but must be written to with a slower process. Once written to, it is used like a ROM, but can be completely erased if necessary.

editor—a program that allows text to be entered into memory. Interactive languages usually have their own editor.

EOF—End Of File.

EOL—End Of Line (of text).

EPROM—Electrically Programmable Read Only Memory. The chip is programmed by voltages higher than normal for computer chips. Once programmed, it is used like ROM, but can be erased by exposure to ultraviolet light.

exclusive OR—see XOR.

execution cycle—a cycle during which a single instruction actually occurs.

expansion interface—a device attached to the computer that allows a greater amount of memory or attachment of other peripherals.

F

fetch cycle—a cycle during which the next instruction to be performed is read from memory.

file—a set of data, specifically arranged, that is treated as a single entity by the software or storage device.

firmware—software that is made semi-permanent by putting it into some type of ROM.

flag—a single bit that is high (set) or low (reset), used to indicate whether or not certain conditions exist or have occured.

flowcharting—a method of graphically displaying program steps, used to develop and define an algorithm before writing the actual code.

FORTRAN—FORmula TRANslator. One of the first high-level languages, written specifically to allow easy entry of mathematical problems.

G

game theory—see von Neumann.

garbage—computer term for useless data.

gate—a circuit that performs a single Boolean function.

GIGO—Garbage In, Garbage Out. One of the rules of computing. If the data going into the computer is bad, the data coming out will be bad also.

graphics—information displayed pictorially as opposed to alphanumerically.

H

half duplex—data can flow in both directions, but not simultaneously. See duplex.

handshaking—a term used in data transfer. Indicates that beside the data lines there are also signal lines so both devices know precisely when to send or receive data. Contrast with buffer.

hangup—a situation where it seems the computer is not listening to you.

hard copy—a printout.

hardware—refers to any physical piece of equipment in a computer system.

hexadecimal—a number system based on sixteen. The decimal digits 0-9 are used along with the alpha characters A-F, which are also recognized as digits.

high—a signal line logic level. The computer senses this level and treats it as a binary 1.

high-level language—a programming code that does not require a knowledge of the CPU structure.

high order—see most significant.

HIT—acronym for Hash Index Table. A section of the directory on a TRS-80 disk.

host computer—the primary computer in a multi-computer or terminal hookup.

human engineering—usually refers to designing hardware and software with ease of use in mind.

I

IC—integrated circuit. See chip.

immediate addressing—the address of the information that an operation is supposed to act upon immediately follows the operation code.

increment—to increase, usually by one. See decrement.

indexed—the information is addressed by a specified value, or by the value in a specified register.

indirect—the address given points to another address, and the second address is where the information actually is.

intelligent terminal—a terminal with a CPU and a certain amount of

memory that can organize the data it receives and thus achieve a high level of handshaking with the host computer.

interactive computing—refers to the appearance of a one-to-one human-computer relationship.

interface—a piece of hardware, specifically designed to hook two other devices together. Usually some software is also required.

interpreter—a piece of system software that executes a program written in a high-level language directly. While useful for interactive computing, this system is too slow for most serious programming. Contrast with compile.

interrupt—a signal that tells the CPU that a task must be done immediately. The registers are pushed to the stack, and a routine for the interrupt is branched to. When finished, the registers are popped from the stack and the main program continues.

I/O—acronym for input/output. Refers to the transfer of data.

J

jack—a socket where wires are connected.

K

K—abbreviation for kilo. In computer terms 1024, in loose terms 1000.

L

least significant—refers to the lowest position digit of a number, or rightmost bit of a byte. In 19963 the 3 is the least significant digit. Opposite of most significant.

LIFO—acronym for Last In First Out. Most CPUs maintain a "stack" of memory that this rule applies to. The last piece of data pushed into the stack is the first piece popped out.

light pen—a device that senses light, interfaced to the computer for the purpose of drawing on the CRT screen.

loop—a set of instructions that executes itself continuously. If the programmer had the presence of mind to provide for a test, the loop is discontinued when the test is met, otherwise it goes on until the machine is shut down.

loop counter—one way to test a loop. The counter is incremented at each pass through the loop. When it reaches a certain value the loop is terminated.

low—a logic signal voltage. The computer senses this as a binary 0.

LSI—acronym for Large Scale Integration. An integrated circuit with a large number of circuits such as a CPU. See chip.

M

machine code—refers to programming instructions that are stored in binary and can be executed directly by the CPU without any compilation, interpretation, or assembly.

machine language—the primary instructions that were designed into the CPU by the manufacturer. These instructions move data between memory and registers, perform simple adding in registers, and allow branching based on values in registers.

macro—a routine that can be separately programmed, given a name, and executed from another program. The macro can perform functions on variables in the program that called it without disturbing anything else and then return control to the calling program.

mainframe—refers to the CPU of a computer. This term is usually confined to larger computers.

memory—the hardware that stores data for use by the CPU. Each piece of data (bit) is represented by some type of electrical charge. Memory can be anything from tiny magnetic doughnuts to bubbles in a fluid. Most microcomputers have chips that contain many microscopic capacitors, each capable of storing a tiny electrical charge.

microprocessor—a CPU on a single chip.

mnemonic—a short, alphanumeric abbreviation used to represent a machine-language code. An assembler will take a program written in these mnemonics and convert it to machine code.

modem—MOdulator/DEModulator. An I/O device that allows communication over telephone lines.

monitor—1. a CRT. 2. a short program that displays the contents of registers and memory locations and allows them to be changed. Monitors can also allow another program to execute one instruction at a time, saving programs and disassembling them.

most significant—refers to the highest value position of a number of the left most bit of a byte. In the number 1923 the 1 is most significant because it represents thousands.

multiplexing—a method allowing several sets of data to be sent at different times over the same bus lines, yet all of the data can be used simultaneously after the final set is received. For example, several LED displays, *each* requiring four data lines, can all be written to with only one group of four data lines.

N

NAND—an acronym for NOT AND. A Boolean logic expression. AND is performed, then NOT is performed to the result.

nanosecond—one billionth of a second.

nesting—putting one loop inside another. Some computers have a limit to the number of loops that can be nested.

NOT—a Boolean operator that reverses outputs (1 becomes 0, 0 becomes 1).

O

object code—all of the machine code that is generated by a compiler or assembler. Once object code is loaded into memory it is called machine code.

octal—refers to the base 8 number system, using digits 0-7.

OEM—Original Equipment Manufacturer.

offset value—a value that can be added to an address. Most addressing modes allow an offset value.

off-the-shelf—a term referring to software. Means the program is general-

ized so that it can be used by a greater number of computer owners, thus it can be mass produced and bought "off-the-shelf."

on-line—a term describing a situation where one computer is connected to another, with full handshake, over a modem line.

OR—a Boolean logic function. If at least one of the lines tested is high (binary 1), the answer is high.

overflow—a situation that occurs when an arithmetic function requires more than the machine is capable of handling. Most computers have a flag so that this condition can be tested.

overlay—a method of decreasing the amount of memory a program uses by allowing sections that are not in use simultaneously to load into the same area of memory. The new routine destroys the first routine, but it can always be loaded again if needed. Usually used in system programs.

oxide—an iron compound coating on tapes and disks that allows them to be magnetized so that they can be read by electrical devices and the information converted back to machine code.

P

page—refers to a 256 (2 to the 8th power) word block of memory. How large a word is depends on the computer. Most micros are 8-bit word machines. The term is important because many chips do special indexed and offset addressing on the page where the program counter is pointing and/or on the first page of memory.

parallel—describes a method of data transfer where each bit of a word has its own data line, and all are transferred simultaneously.

parameter—a variable or constant that can be defined by the user and usually has a default value.

parity—a method of checking accuracy. The parity is found by adding all the bits of a word together. If the answer is even the parity is 0 or even. If odd, the parity is 1 or odd. The bit sometimes replaces the most significant bit and usually sets a flag.

PC board—stands for Printed Circuit board. A piece of plastic board with

lines of a conductive material deposited on it to connect the components. The lines act like wires. These can be manufactured quickly and are easy to assemble the components on.

peripheral—any piece of hardware that is not a basic part of the computer.

PILOT—a simple language for handling English sentences and strings of alphanumeric characters.

PL/1—a programming language used by very large computers. It incorporates most of the better features from other programming languages.

plotter—a device that can draw graphs and curves controlled by the computer through an interface.

port—a single addressable channel used for communications.

PROM—Programmable Read Only Memory. A memory device that is written to once, and from then on acts like a ROM.

pseudo code—a mnemonic used by assemblers that is not a command to the CPU, but a command to the assembler itself.

R

RAM—an acronym for Random Access Memory. Memory that can be written to or read from. It is addressed by the address bus.

real time clock—a clock in the sense that we normally think of one, interfaced to the computer.

record—a file is divided into records, each of which is organized in the same manner.

register—a memory location used by the CPU and not addressed by the address bus. It cannot be used by the programmer.

relative addressing—an address that is dependent upon where the program counter is presently pointing.

ROM—an acronym for Read Only Memory. Memory that is addressed by the bus, but can only be read from. If you tell the CPU to write to it, the

machine will try, but the data is not remembered.

RPG—an acronym for Report Program Generator. A language for business that primarily reads data from cards and prints reports containing that data.

RS-232—an interface that converts parallel data to serial data for communications purposes. The output is universally standard.

S

semiconductor—a compound that can be made to vary its resistance to electricity by mixing it differently. Layers of this material can be used to make circuits that do the same things tubes do, but using much less electricty. Transistors and integrated circuits are made from semiconductive material and called semiconductors.

serial—a way of sending data, one bit at a time, between two devices. The bits are rejoined into bytes by the receiving device. Contrast with parallel.

sign bit—sometimes the most significant bit is used to indicate the sign of the number it represents. 1 is negative (−) and 0 is positive (+).

simulator—a computer that is programmed to mimic the action and functions of another piece of machinery, usually for training purposes. A computer is usually employed because it is cheaper to have the computer simulate these actions than to use the real thing. Airplane and power plant trainers are excellent examples.

software—refers to the programs that can be run on a computer.

source program—the program written in a language or mnemonics that is converted to machine code. The source program as well as the object code generated from it can be saved in mass storage devices.

stack—an area of memory used by the CPU and the programmer particularly for storage of register values during interrupt routines. See LIFO.

status register—the register that contains the status flags set and tested by the CPU operations.

stepper motor—a special motor in a disk drive that moves the read/write head a specific distance each time power is applied. That distance defines the tracks on a disk.

subroutine—a routine within a program that ends with an instruction to return program flow to where it was before the routine began. This routine is used many times from many different places in the program, and the subroutine allows you to write the code for that routine only once. Similar to a macro.

syntax—the term is used exactly as it is used in English composition. Every language has its own syntax.

system software—software that the computer must have loaded and running to work properly.

T

table—an ordered collection of variables and/or values, indexed in such a way that finding a particular one can be done quickly.

text editor—see word processor.

time-sharing—refers to systems which allow several people to use the computer at the same time.

track—a concentric area on a disk where data is stored in microscopic magnetized areas.

TTL—Transistor-Transistor Logic. Means that the electrical values for logic highs and lows fall within the values necessary to run transistors. See semiconductor.

U

utility—a program designed to aid the programmer in developing other software.

V

variable—a labeled entity that can take on any value.

von Neumann, John (1903-1957)—Mathemetician. Put the concept of games, winning strategy, and different types of games into mathematical

formulae. Also advanced the concept of storing the program in memory as opposed to having it on tape.

W

word—in computing it refers to a number of bits that are in a parallel format. If the CPU works with 8 bits then the word length is 8 bits. Common word sizes are 4, 8, 12, 16, and 32. Some are as large as 128 bits.

word processor—a computer system dedicated to editing text and printing it in various controllable formats. See editor.

write—to store in memory or on a mass storage device.

X

XOR—a Boolean function. Acronym for eXclusive OR. Similar to OR but answer is high (1) if and only if one line is high.

Z

zero page—refers to the first page of memory.

INDEX

INDEX

INDEX COMPILED BY NAN MCCARTHY

Encyclopedia Loader™

The editors of Wayne Green Books want to help you use the programs in your **Encyclopedia for the TRS-80**. So to help you maximize the use of your microcomputing time, we created **Encyclopedia Loader™**.

By a special arrangement with Instant Software™, Wayne Green Books can now provide you with selected programs contained in each volume of the **Encyclopedia for the TRS-80** on a special series of cassettes called **Encyclopedia Loader™**. Your encyclopedia provides the essential documentation but now you'll be able to load the programs instantly. Each volume of the Encyclopedia will have a loader available.

With **Encyclopedia Loader™** you'll save hours of keyboard time and eliminate the aggravating search for typos. **Encyclopedia Loader™** for Volume 3 will contain the programs in the following articles:

Flex/Form
Algebra Tutor
Supermaze
Micro Basketball
The Great Girl Scout Cookie Caper
Two Energy Savers
CISAB—Backwards BASIC
Spool and Despool.

Encyclopedia Loader™ for Volume 1

EL8001 $14.95

(plus $1.50 postage & handling)

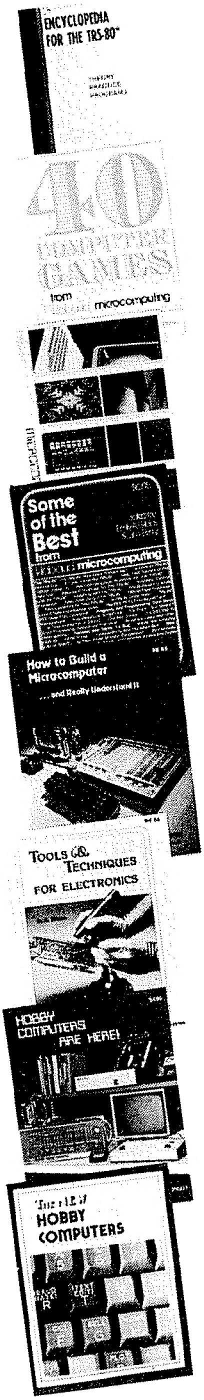
Encyclopedia Loader™ for Volume 2

EL8002 $14.95

(plus $1.50 postage & handling)

Mail your order to "Encyclopedia Loader Sales," Wayne Green Books, Pine Street, Peterborough, NH 03458 or call **1-800-258-5473.**

WAYNE GREEN BOOKS



Encyclopedia for The TRS-80*—A ten-volume series to be issued every two months starting July 1981. The Encyclopedia contains the most up-to-date information on how to use your TRS-80*.

40 Computer Games from Kilobaud Microcomputing—Games in nine different categories for large and small systems, including a section on calculator games.

Understanding and Programming Microcomputers—A well-structured introductory text on the hardware and software aspects of microcomputing.

Some of the Best from Kilobaud Microcomputing—A collection of articles focusing on programming techniques and hardcore hardware construction projects.

How to Build a Microcomputer and Really Understand It—A technical manual and programming guide that takes the hobbyist step-by-step through the design, construction, testing and debugging of a complete microcomputer system (6502 chip).

Tools and Techniques for Electronics—Describes the safe and correct ways to use basic and specialized tools for electronic projects as well as specialized metal working tools and the chemical aids which are used in repair shops.

Hobby Computers Are Here—The fundamentals on how computers work—explaining the circuits and the basics of programming plus a couple of TVT construction projects.

The New Hobby Computers—Contains introductory articles on such subjects as large scale integration, how to choose a microprocessor chip, and introduction to programming, computer arithmetic, etc.

To order call Toll Free 800-258-5473.

*TRS-80 is a trademark of Radio Shack Division of Tandy Corp.

The real value of your computer lies in your ability to use it. The capabilities of the TRS-80* are incredible if you have the information which will help you get the most from it. Little of this information is available in your instruction books.

The *Encyclopedia for the TRS-80* will teach you how to get the most from your computer. In addition to a wealth of programs which are ready for you to use, reviews of accessories and commercially available programs, you will also learn how to write your own programs or even modify commercial programs for your own specific use.

The *Encyclopedia* is packed with practical information, written and edited for the average TRS-80 owner, not the computer scientist. You will find it interesting and valuable.

Wayne Green
Publisher

ISBN 0-88006-032-8